ISSN 0280-5316
ISRN LUTFD2/TFRT--5794--SE

Automatic Calibration of Vehicle Models

Henrik Hultgren
Henrik Jonasson

Department of Automatic Control
Lund University
June 2007

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
June 2007

Document Number
ISRNLUTFD2/TFRT--5794--SE

Author(s)
Henrik Hultgren and Henrik Jonasson

Supervisor
Magnus Gäfvert and Johan Andreasson at Modelon in Lund. Johan Åkesson at Automatic Control in Lund. Anders Rantzer Automatic Control in Lund (examiner)

Sponsoring organization

Title and subtitle
Automatic Calibration of Vehicle Models (Automatisk kalibrering av fordons modeller)

Abstract
In this thesis simple vehicle models are compared to more advanced models. The parameters in the simple models have to be chosen in such a way, that the model behaviour is as close to the advanced model as possible. This parameter optimisation is done with help of The Optimica Compiler, wich generates AMPL files that solves the problems. To run a successful parameter estimation there is a need for polynomial interpolation of the measurements from the advanced model. This is done with the spline toolbox **Splines1.1** for Modelica.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
93

Recipient's notes

Security classification

http://www.control.lth.se/publications/

ACKNOWLEDGEMENTS

Thanks to Modelon AB, especially Magnus Gäfvert and Johan Andreasson for the opportunity and help to carry out this thesis. For excellent support and guidance we would also like to give thanks to Johan Åkesson, PhD student at the department of automatic control LTH.

Henrik Hultgren & Henrik Jonasson

Lund June 2007

Contents

1 Introduction **1**
1.1 Background 1
1.2 Purpose 1
1.3 Modelon 2
1.3.1 Vehicle Dynamics Library 2
1.4 Outline 2

2 Curve Fitting with Splines **5**
2.1 Piecewise Polynomials 5
2.2 B-splines 6
2.3 FORTRAN Spline Functions 8
2.3.1 Redeclaration of Variable Types 8
2.3.2 FORTRAN90 8
2.3.3 C-wrappers 8
2.4 Splines in Modelica 10
2.5 Modification of FORTRAN Subroutine `interv` 10
2.6 Spline Tools In Modelica 11

3 Vehicle Models **13**
3.1 Model Overview 13
3.1.1 One Track Model with Linear and Nonlinear Tires . . 13
3.1.2 Two Track Model with Nonlinear Tires 14
3.1.3 Two Track Model with Nonlinear Tyres and Roll 14
3.2 Modelica 14
3.2.1 DYMOLA 15
3.3 Vehicle Models 15
3.3.1 One Track Model with Linear Tyres 15
3.3.2 One Track Model with Nonlinear Tyres 16
3.3.3 Behaviour of linear and nonlinear tyres 16
3.3.4 Two Track Model with Linear Tyres 17
3.3.5 Two Track with Roll and Load Distribution (Linear Tyres) 19
3.3.6 Two Track with Roll, Load Distribution and Magic Formula Tyres 20
3.4 Modelling Tyres 20
3.4.1 Slip Angle 20
3.4.2 Linear Tyre Model 22

3.4.3 Magic Formula for lateral slip 22
3.4.4 Longitudinal Slip 23
3.4.5 Magic Formula for Combined Slip 24
3.4.6 Brush Model 24

4 Optimisation 29
4.1 The Optimica Compiler 29
4.2 Dynamic Optimisation 30
4.2.1 Orthogonal Collocation 30
4.3 AMPL and IPOPT 30
4.4 Intermediate Spline Solutions 31
4.4.1 Matlab Splines 31
4.4.2 AMPL Splines 32
4.5 Parameter Optimisation 32
4.5.1 Servo Optimisation 33

5 Parameter Optimisation in Vehicle Models 37
5.1 One Track Model 37
5.1.1 One Track Model with Linear Tyres 37
5.1.2 One Track Model with Nonlinear Tyres 38
5.1.3 Optimisation of Nonlinear Tyre Parameters 41
5.2 Optimisation and VDL 42
5.2.1 Vehicle Manoeuvres 42
5.2.2 Parameter Optimisation Procedure 43
5.2.3 VDL model 44

6 Conclusion 49
6.1 Spline Toolbox 49
6.2 Optimisation 49
6.2.1 General optimisation 49
6.2.2 Vehicle Parameter Optimisation 50
6.3 Future Work 50
6.3.1 Splines1.1 50
6.3.2 Vehicle Parameter Optimisation 50

Appendix A Spline Tools in Modelica 53
A.1 Examples 53
A.2 Spline construction 57

Appendix B Vehicle Models 65
B.1 One Track with Linear Tyres 65
B.2 Two Track Linear Tyres 66
B.3 Two Track Magic Formula and Load Distribution 68

Appendix C AMPL Spline Code 73
C.1 funcadd.c 73
C.2 Roll.run 80
C.3 Roll.Cost.mod 80

List of Figures

2.1 Different polynomial interpolations 6
2.2 The four first B-splines 7
2.3 Principle of modified interv function 11

3.1 One track model. 16
3.2 Lateral tyre forces in the linear region 17
3.3 Lateral tyre forces in the nonlinear region 18
3.4 Lateral acceleration in the nonlinear region 19
3.5 Two track model. 20
3.6 Slip angle is defined as the difference between the wheel centre velocity and the wheel plane. 21
3.7 Characteristic of tyres. 22
3.8 Effective rolling radius. 23
3.9 Extract from Figure 3.10 showing how to calculate lateral displacement for a bristle. 24
3.10 The brush model for lateral force 25
3.11 Longitudinal slip for the brush model. 27

4.1 Flowchart over the steps involved in running an optimisation problem 29
4.2 Splines were created with a MATLAB script. 31
4.3 Steps involved to run an AMPL spline 32
4.4 Process and model are excited with the same input and the difference in output will be minimised. 33
4.5 The model structure in DYMOLA. 33
4.6 Comparison between the model and the real servo 34
4.7 The dynamics is modelled in Modelica and the optimisation specifications are written in Optimica code 35
4.8 Parameter estimation of spring-mass system 36

5.1 Original, simulated initial guess and optimised a_y 38
5.2 Original, simulated initial guess and optimised a_y for linear model versus nonlinear 39
5.3 Global model positions after optimisation, the tyres are in the linear region 40
5.4 Original, simulated initial guess and optimised a_y for linear model versus nonlinear in the nonlinear region 41

5.5 Global model positions for linear model versus nonlinear in the nonlinear region after optimisation 42
5.6 Original, simulated initial guess and optimised a_y for optimisation of Magic Formula parameters 43
5.7 Original, simulated initial guess and optimised $\ddot{\Psi}$ for optimisation of Magic Formula parameters 44
5.8 Comparison of lateral acceleration between VDL-, initial guess- and optimised model for a steer ramp. 46
5.9 Comparison of yaw rate between VDL-, initial guess- and optimised model for a steer ramp. 46
5.10 Comparison of lateral acceleration between VDL-, initial guess- and optimised model for a lane change. 47
5.11 Comparison of yaw rate between VDL-, initial guess- and optimised model for a lane change. 47

A.1 The structure of the spline package 54
A.2 There are six examples for demonstration purpose included in the package. 55
A.3 To plot results a special command is necessary. 55
A.4 A complete test of all examples 56

List of Tables

5.1 Parameter optimisation results for One Track Model. 38
5.2 Parameter optimisation results for linear One Track Model versus nonlinear when the tyres are in the linear region. . . . 40
5.3 Parameter optimisation results for linear One Track Model versus nonlinear when the tyres are in the nonlinear region. . 41
5.4 Parameters in magic formula optimised. 42
5.5 Parameters optimised in VDL model (steer ramp). 45
5.6 Parameters optimised in VDL model (lane change). 45

6.1 Suitable manoeuvres and cost function states for finding different types of parameters 50

1

INTRODUCTION

This thesis is carried out in cooperation with Modelon AB and the Department of Automatic Control, Lund University.

1.1 Background

A detailed model that describes the movements of a vehicle may be very complex and may contain tens of thousand equations. It is difficult to understand every detail in such an advanced model. Problems also occur due to the model complexity when designing control systems. If less complex models could imitate the behaviour of more advanced ones it would be of interest when determining parameters and conditions.

1.2 Purpose

The aim of this master thesis is to construct reduced car models such as one track models or simple two track models of low complexity that are nearly as good, in comparison to the complex ones during certain conditions, but with fewer design parameters and less complexity. In these small models there are parameters that have to be tuned in order to make the model perform in an *optimal way*. Optimal meaning a minimisation of a specific cost function. In the cost function different state derivations between the real model and the reduced model are punished.

A new tool for optimisation of Modelica models called **T**he **O**ptimica **C**ompiler or TOC [Åkesson Johan, 2007] is developed by Johan Åkesson at the Department of Automatic Control at Lund University. This is utilised during the parameter optimisation process.

In order to use the data from the advanced models, which will be considered the truth or optimum, there has to be a polynomial interpolation between the discrete data points. This due to that the optimisation tool

needs a continuous and two times differentiable function as input. Polynomial interpolation could be done in many ways but in this thesis splines will be used. For this purpose a special spline toolbox for DYMOLA has been developed.

1.3 Modelon

Modelon AB is a company that provides advanced models in Modelica for automotive, aerospace and process industries. Modelon has developed several libraries for DYMOLA, among them, the Vehicle Dynamics Library (VDL), is of interest for this thesis [Modelon, 2007].

1.3.1 Vehicle Dynamics Library

This library is a tool for simulation and analysis of the dynamics during handling manoeuvres of a vehicle. It is based on the Modelica programming language. VDL is a drag-and-drop system that is easy to use, it is also possible to edit the Modelica source code for more advanced features.

Handling behaviour is the primary aim of this library, but other vehicle properties are also possible to study. It is feasible to involve other types of components like electronics, pneumatics and hydraulics. Also detailed models from other model libraries such as Transmission- (Ricardo, UK), PowerTrain- (DLR, Germany) and SmartElectricDrives-library (Arsenal Research, Austria) can be used. This makes VDL to a very flexible tool. Other examples on features are the test rigs which make it possible to isolate vehicle behaviour and wheel test rigs for validation of models against measurements. 3D roads are constructed with tabular data which makes it possible for the user to define own surfaces. The possibility to use VDL models in Simulink makes it practical to use Simulink as a design tool of controllers and then try it in the VDL model.

The purpose with VDL in this thesis is to compare the reduced models against those in VDL and estimate parameters in an optimal manner.

1.4 Outline

The thesis consists of two major parts, the spline part and the optimisation part. The optimisation part is divided in two areas, modelling of vehicle

dynamics and parameter optimisation.

Chapter 2 considers the polynomial reconstruction of the simulation data sets. In Chapter 3, simple one and two track models are constructed. Chapter 4 discusses the optimisation process and how to optimise parameters. To illustrate the optimisation process a simple optimisation experiment with two masses and a spring is also performed.

Further optimisation on vehicle models is done in Chapter 5. The last chapter contains a conclusion of the thesis, analyses and discussions on the obtained results.

The functions in the spline package are described in Appendix A and the Modelica source code for the models is in Appendix B. Source code for the intermediate solution with AMPL spline are in Appendix C.

2

Curve Fitting with Splines

To make the parameter optimisation possible, the discrete data points from the simulated advanced model have to be transformed into a continuous differentiable function. This can be done by using some sort of *interpolation* between the points, see Figure 2.1. In this thesis two types of constructions have been used.

- Piecewise polynomials
- B-spline

As will be described later on both types have their advantages and disadvantages. For easy handling and convenience, the aim was to construct a toolbox containing spline tools similar to the one in MATLAB [de Boor, 2007]. The *spline-toolbox* that has been created in DYMOLA has similar features as the MATLAB *spline-toolbox*. The MATLAB *spline-toolbox* has also been used to validate spline results created in DYMOLA.

The starting point was the PPPack which contains the spline algorithms written in FORTRAN code [de Boor, 1978]. These subroutines will be called from DYMOLA by an external function call, which causes some problems as described below. Different approaches have been made to overcome communication problems due to different variable types in FORTRAN and Modelica.

2.1 Piecewise Polynomials

It is possible to approximate a function f with a polynomial as long as the function is sufficiently smooth and the approximated interval is rather small. If the interval is large the polynomial has to be of a high order and with increasing length of the interval the order eventually becomes unacceptably large. Also if the function to be approximated behaves badly somewhere

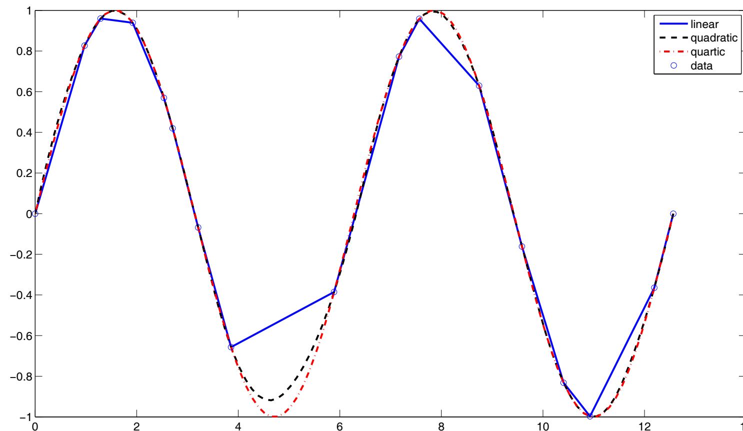


Figure 2.1: *Different polynomial interpolations*

in the interval of the approximation, then the approximation is poor everywhere. To avoid large oscillations it is common to use so called piecewise polynomials. The idea is that large intervals are divided into smaller intervals and a polynomial of low degree is fitted in each subinterval, hence the name piecewise polynomial. The main advantage of this pp-interpolation is that a large number of data points can be fitted with a polynomial with low degree. The simplest polynomial that can be fitted between two points is a straight line, which is called piecewise linear interpolation. In such a piecewise polynomial there are different polynomials that are used in the different intervals. The points where one polynomial changes to another, are called *knots* or *breakpoints*. Assuming the breakpoint sequence consists of strictly increasing points

$$\tau_i < \tau_2 < \ldots < \tau_{l+1}$$

then the *breakpoints*, *coefficients*, *number of polynomial pieces* and the *order* represent the *pp-form*. An advantage of the pp-form is its efficiency to evaluate. However, when it comes to construction of piecewise-polynomials the efficiency is lower than for the B-form.

2.2 B-splines

The *B-spline* is the standard representing format when constructing splines due to the possibility to enforce smoothness requirements across breaks. A B-spline is a weighted sum of basis functions hence the name *Basis-spline* [Heath, 2002].

$$\sum_{j=1}^{n} B_j^k a_j$$

where B_j^k is a is piecewise polynomial of degree $< k$ with knots $t_j, \ldots, t_{j+k}$ and a_j are the B-spline coefficients. The first B-spline of degree 0 is defined

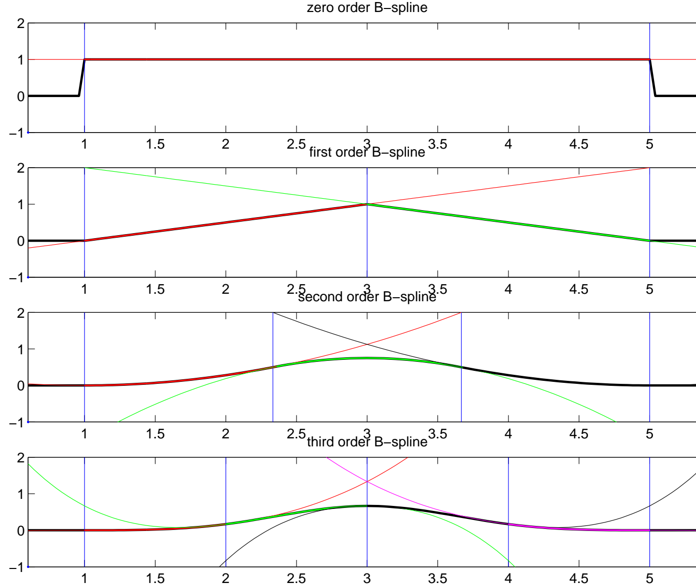


Figure 2.2: *The four first B-splines. Vertical thin lines denotes placement of knots. As can be seen the first B-spline consists of only one polynomial piece, the second consist of two pieces etc.*

as

$$B_i^0(t) = \begin{cases} 1 & \text{if} \quad t_i < t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

The following B-splines are calculated recursively.

$$B_i^k(t) = v_i^k(t)B_i^{k-1}(t) + v_{i+1}^k(t)B_{i+1}^{k-1} \tag{2.2}$$

where

$$v_i^k(t) = \frac{t - t_i}{t_{i+k} - t_i} \tag{2.3}$$

The four first B-splines are depicted in Figure 2.2. A B-spline consists of a *knot-sequence*, *B-spline coefficients* and the *order*. The difference between knots and breakpoints is the non-strictly increasing requirement, which allows repeated knots i.e. the knots could have *multiplicities*.

$$t_1 \leq t_2 \leq \ldots \leq t_{i+k}$$

The relation between knots, order and smoothness conditions is

knot multiplicity + condition multiplicity = order.

Smoothness condition is a figure that describes the number of continuous derivatives. The knot multiplicity affect the smoothness condition so that higher multiplicity results in fewer continuous derivatives. For a B-spline of order three, the meaning of the smoothness condition is

- smoothness condition 2 means a continuous function and a continuous first derivative ($1 + 2 = 3$ in relation above)
- smoothness condition 1 means only a continuous function ($2 + 1 = 3$ in relation above)

the relation is the same for higher order B-splines. A B-spline with the above knot-sequence is zero outside the interval $t_1 \dots t_{i+k}$. As discussed above the main advantage compared to the pp-form is the efficiency to construct splines.

2.3 FORTRAN Spline Functions

Carl de Boor has written several FORTRAN-routines that construct piecewise polynomials and B-splines. In addition, there are routines for calculating the jth derivative value in a certain point, convert B-spline to pp, and treat the breaks and knots. These can be obtained from the public available PPPack (`http://netlib.org/pppack/index.html`).

2.3.1 Redeclaration of Variable Types

These original FORTRAN-routines are written in FORTRAN77 and the aim was to be able to use them from DYMOLA without any change. A problem that occurred initially was that DYMOLA calls external FORTRAN code with double precision (64 bits) and the FORTRAN-routines uses single precision (32 bits). The first attempt to overcome this obstacle was to manually change the declarations in the routines. This worked for some of the routines but others contained intrinsic calls that required single precision arguments. If the approach would have been successful the idea was to write a script that simply changed all the declarations from *real* to *double precision* but the FORTRAN compiler is very strict with column layout and the longer declaration sometimes caused the text to exceed the limits which caused an error. This could be overcome by manually adjusting the code, however this was not desirable. The idea was that anyone should be able to use the spline package without changes in the original PPPack.

2.3.2 FORTRAN90

The next approach was to use pppack.f90 a PPPack that is rewritten in FORTRAN90 (also a public code package) and it had the correct variable type from the beginning, *double precision*. This appeared to work at first but then it was discovered that the source code was under development and not bug free. Since this code is changing, and was changing during the work process, it was considered unstable and was abandoned. The source code can be found on (`http://people.scs.fsu.edu/~burkardt/f_src/pppack/pppack.html`).

2.3.3 C-wrappers

Eventually it was decided to use the original FORTRAN code unchanged since it came from a reliable source and appeared thoroughly tested. In order

to do this a layer between DYMOLA and FORTRAN had to be created, i.e. a wrapper. These were to be written in C for simplicity and compliance with DYMOLA. The wrappers had to contain all the conversions between double and single precision. All arrays are declared and created in DYMOLA to utilise the memory maximally, including work arrays for C and FORTRAN. In the respective wrappers n dimensional arrays are converted from row convention in C to column convention in FORTRAN and vice verse. Below is an example on a wrapper structure

```
extern void cubspl_(const float*, const float*, const int*,
                    const int*, const int* );

int cubsplwc(double* tau, double* bc, int n, int ibcbeg, int ibcend,
             float* tauW, float* bcW) {

  int i, j;

  for (i=0;i<n;++i){
    tauW[i] = (float)tau[i];
  }

  for (i=0;i<n;++i){
    bcW[i*4] = (float)bc[i]; //data
  }

  //conditions
  bcW[1] = (float)bc[n];
  bcW[4*n-3] = (float)bc[2*n-1];

  /*### call FORTRAN function ###*/
  cubspl_(tauW, bcW, &n, &ibcbeg, &ibcend);

  /* Revert, Dymola (c) Row - FORTRAN Column*/
  for(j=0;j<n;++j){
    for(i=0;i<4;++i){
      bc[i*n+j]=(double)bcW[i+j*4];
    }
  }

  return splineInit();
}
```

All wrappers are written in similar fashion.

- Conversion from double precision to single precisions is done with loops in the wrapper. Due to the different matrix row and colon notations in C and FORTRAN it is also necessary to transpose the matrices.
- Call the external FORTRAN subroutine.
- Transpose once again and convert back to double precision.

The last expression `return splineInit()` gives the spline a individual identification number, as explained below.

An interesting aspect to consider is the time that each C-wrapper consumes. Measurements on wrappers have been done and the results indicates that the time lag that wrappers contributes are very small, less than μs. This depends of course on the matrix dimensions, in the test above the size of the data inputs were 2000×1.

2.4 Splines in Modelica

When the C-wrapper were written so they could pass the arrays, matrices and values to the FORTRAN subroutines the challenge was to write Modelica functions in DYMOLA. In these functions external calls to the C-wrappers made it possible to use the FORTRAN routines. The functions written in Modelica are split into two levels. The purpose of the first level functions is to construct values, arrays and matrices that the higher level functions not necessarily have to see. In this level all the work arrays that the wrappers and FORTRAN need are created. The first layer is named *PPPack* and the second *Splines*.

2.5 Modification of FORTRAN Subroutine `interv`

When splines are to be evaluated a function `ppval` or `bvalu` has to be called. Inside these two subroutines there is a call for a function `interv` written in FORTRAN. Its commission is to find in which interval the point to be evaluated is located in.

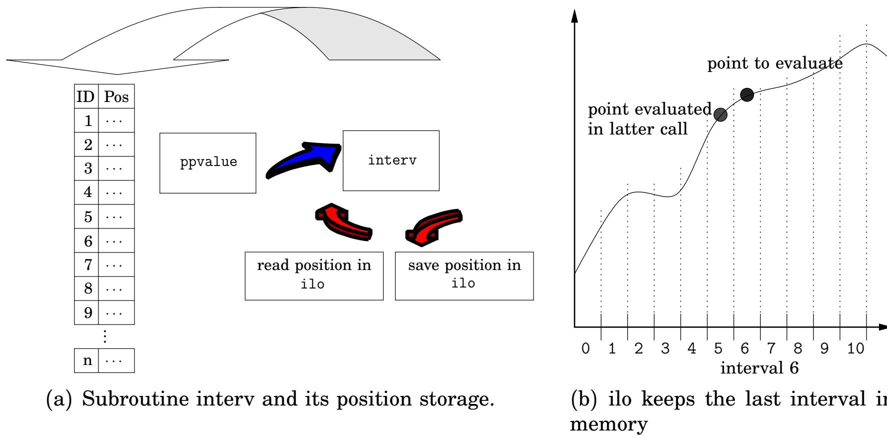
In `interv` a local variable `ilo` keeps the position from the last call in memory. When it is called again it does not have to search through the whole data set to find the right evaluation point, it just looks where it was in the last call. It is an advantage to do so when the evaluation functions are to be called repeatedly. This original solution has a problem when different splines with different data sets are calling `ppval` or `bvalu` repeatedly, then the locally stored variable is not valid for more than one spline. It will mix up the intervals and the previous call do not necessarily contain any relevant information to the current call.

To overcome this, specific Id's for every spline is introduced. Every spline has its own interval memory stored in a static array. When constructing a spline it receives an ID-number, this number corresponds to a interval position stored in the array, see Figure 2.3. With this solution every spline has its own interval position stored and the intervals are kept separate. This results in a faster evaluation when large data sets and multiple splines are to be handled. The updated version of `ppval` as described above is `ppval2`.

However today's fast computers working with GHz processors make this interval search fast. An experiment made with two different splines constructed with `ppcub` with 2000 respective 5 breakpoints, calling `ppval` and `ppval2` repeatedly 50 000 times in different points showed no significant improvements in time. There was however an improvement in the number of iterations that were made between the two versions.

- `ppval` needed 824 798 iterations
- `ppval2` needed 2001 iterations

Assumed that `ppval` needs 1000 instructions more than `ppval2`, this is a high figure, in each function call to be able to find the right interval. The



(a) Subroutine interv and its position storage.

(b) ilo keeps the last interval in memory

Figure 2.3: *The lower interval is stored in ilo which is saved in a global vector with different intervals for different splines. This makes it possible for a spline to keep its last position even though other splines are called in between*

computer works with approximately 2.5 GHz.

$$\frac{1}{2.5\text{ GHz}}(1000\text{ inst})\frac{824798}{2001} = 1,65e^{-4}s$$

These calculations indicate that the speed of the computer is so high that 400 times more iterations, does not make a large difference. However this leads to a more efficient handling of the interval search in `interv` as the results indicate. The result depends on usage, larger array dimensions and multiple splines will give a larger improvement.

2.6 Spline Tools In Modelica

All C wrappers, Modelica wrappers and FORTRAN subroutines are collected and linked together into a DYMOLA package called **Splines 1.1**. It contains several spline construction tools for both piecewise polynomial and B-spline. Also included are tools for work on knots and breakpoints. A few examples included demonstrate the functions of the splines.

To use above described tools, the complete package **Splines 1.1** is needed. It contains the DYMOLA model, wrappers, instructions for how to use and install. For complete functionality most of FORTRAN subroutines in public PPPack have to be downloaded from the Internet, which ones can be seen in README.TXT. **Splines 1.1** only contains the modified versions of `ppval`, `bval` and `interv` located in the folder *src/*.

For more details about functions and the structure of **Splines 1.1**, see manual in Appendix A.

3

Vehicle Models

The models created and presented in this chapter ranges from simple, with few degrees of freedom (DOF), to more advanced with more DOFs. These models are however still quite simple compared to models in VDL. It is desired not to use a more advanced model than required to complete a certain manoeuvre.

3.1 Model Overview

The models that have been constructed are introduced in Sections 3.1.1-3.1.3 with their important input and output signals, area of interest and advantages versus disadvantages. The outputs are states that are interesting to observe, but they are not all the states available. More detailed descriptions are found in Section 3.3.

3.1.1 One Track Model with Linear and Nonlinear Tires

The least complex vehicle model is the One Track with linear tyres.

DOFs 2-3

Inputs Steering angle (δ), Longitudinal velocity (v_x) / acceleration (a_x) / force (f_x)

Outputs Yaw angle velocity ($\dot{\psi}$), lateral velocity (v_y)

Areas of interest Steady state manoeuvres and references to, for example, Electronic Stability Program (ESP)

Advantages Easy to calibrate for the linear case

Disadvantages Does not handle transients, does not handle individual tyre traction and braking

3.1.2 Two Track Model with Nonlinear Tires

This model has four wheels and the tyres are nonlinear, no roll freedom is considered. A load distribution is however included in the model.

DOFs 3

Inputs Steering angle (δ), Longitudinal force (f_i)

Outputs Yaw angle velocity ($\dot{\psi}$), lateral acceleration (v_y)

Areas of interest Steady state manoeuvres and references to, for example, Electronic Stability Program (ESP)

Advantages Individual tyre braking or acceleration, nonlinear load distribution

Disadvantages Does not describe the roll angle (φ)

3.1.3 Two Track Model with Nonlinear Tyres and Roll

The most advanced model with roll dynamics introduced.

DOFs 5

Inputs Steering angle (δ), Longitudinal velocity (v_x) / acceleration (a_x) / force (f_x)

Outputs Roll and pitch

Areas of interest Transient manoeuvres

Advantages Contains most of the dynamics needed

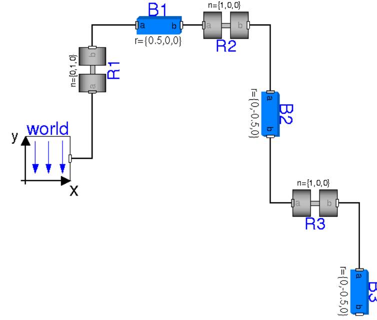
Disadvantages Difficult to find the roll axis, toe and camber are not included, more parameters to tune

3.2 Modelica

All vehicle models that were used during simulations were implemented in Modelica code. Modelica is a free object orientated language suited for modelling complex physical systems. A nice feature in Modelica is that it is quite simple to describe physical systems, one only has to specify the differential equations for the system. Single variable solving is not necessary, this is all done in the compiler. To solve problems with Modelica, a modelling and simulation environment is needed. In this thesis a program called DYMOLA was used.

3.2.1 Dymola

DYMOLA is a program created by the company Dynasim. It has a graphical editor where built in components can be connected in a drag-and-drop manner. The components are mechanical, electrical and thermal. There are also other types of blocks like maths functions and different source blocks etc. Examples of mechanical blocks are inertias, springs and rods. In the graphical editor models are built by differential equations describing their behaviour. To build a model, components that corresponds to the real systems are connected. The DYMOLA model reflects how it looks in reality, see the DYMOLA example *furuta pendulum* to the right. It consists of three joints denoted R and three bodies B which are the arms of the pendulum.



When vehicle models were constructed the graphical editor was not used. Instead force and moment equations were written directly in the text editor.

3.3 Vehicle Models

This section has a more detailed explanation of how the models are constructed. It starts with the One Track model with linear tyres and ends with the Two Track model with nonlinear tyres, pitch and roll. The Modelica code for vehicle models created and used can be found in Appendix B

3.3.1 One Track Model with Linear Tyres

The simplest car model is the One Track model, or the bicycle model, with linear tyres. It is modelled such that the front tyres are combined to one and the rear tyres are combined to one. These are placed in the centre, i.e. the y-coordinate equals zero. Effects such as pitch and roll are discounted when setting the z-coordinate to zero. This gives a model that is easy to calibrate for the linear case and is usable for steady state manoeuvres. It does not handle transients or individual breaking. See Figure 3.1.

The equations needed to put together the model are the force equations and the moment equation:

$$\uparrow \quad m(\dot{v}_x - \dot{\Psi} v_y) = -F_{12}\sin(\delta) \tag{3.1}$$

$$\rightarrow \quad m(\dot{v}_y + \dot{\Psi} v_x) = F_{34} + F_{12}\cos(\delta) \tag{3.2}$$

$$\circlearrowleft \quad J_z\ddot{\Psi} = fF_{12}\cos(\delta) - bF_{34} \tag{3.3}$$

where equation (3.1) are the forces acting in the y-direction, equation (3.2) are the forces acting in the x-direction and equation (3.3) are the moments of inertia around the z-axis through the centre of gravity. The slip angles front (α_{12}) and rear (α_{34}) are calculated according to equations (3.4 - 3.5).

$$\alpha_{12} = \arctan\left(\frac{v_y + \dot{\psi} f}{v_x}\right) - \delta \tag{3.4}$$

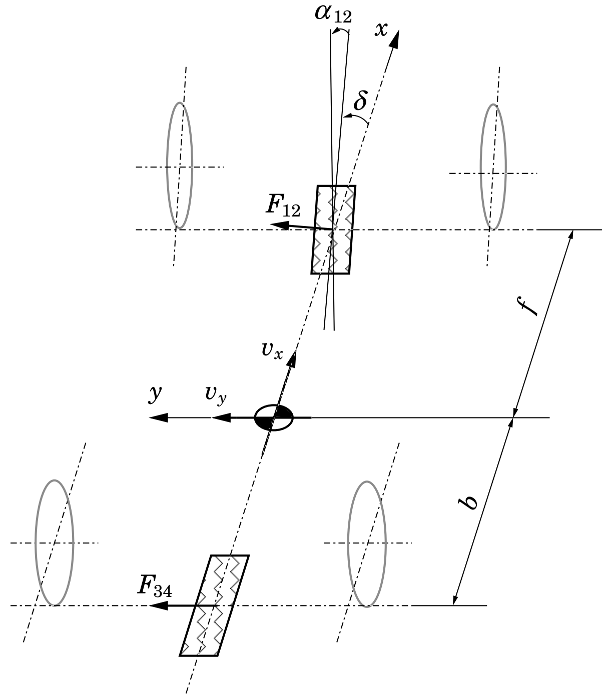


Figure 3.1: *One track model.*

$$\alpha_{34} = \arctan\left(\frac{v_y - \dot{\psi} f}{v_x}\right) \tag{3.5}$$

where δ is the steering angle.

These combined with the force equation for the linear tyre, equation (3.23), and applying the rules for derivation of vectors according to (3.6) are enough to assemble the one track model.

$$\frac{d\overline{f}}{dt} = \frac{\partial \overline{f}}{\partial t} + \overline{\omega} \times \overline{f} \tag{3.6}$$

3.3.2 One Track Model with Nonlinear Tyres

To extend the above One Track model non linear tyres are introduced. The model for the tyres is the Magic Formula equation (3.25).

3.3.3 Behaviour of linear and nonlinear tyres

To verify that the linear and nonlinear tyres behave similar in the linear region and that they differ in the nonlinear region, two simulations were done. See section 3.4 for description of linear and nonlinear tyres. The first simulation demonstrates the linear region. Both models had the same inputs; a steady longitudinal velocity (v_x) of $10m/s$ and a steering angle (δ) that increased from 0-2 rad/s over the simulation period 30 seconds. As can be seen in Figure 3.2 the lateral forces (f_y) are very similar for both models.

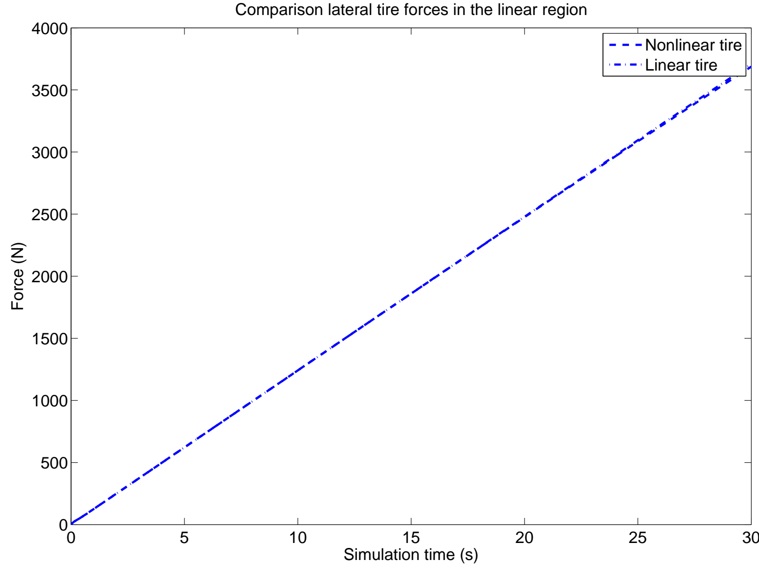


Figure 3.2: *Lateral tyre forces in the linear region*

To make the models enter the nonlinear region a larger longitudinal velocity of $20m/s$ was used. This made the nonlinear tyre reach its top value, which is the maximum lateral value, while the linear tyre behaved as before, see Figure 3.3. The effects of the nonlinearity can be seen in states in the model, for example the lateral acceleration (a_y). In Figure 3.4 it can be seen that the lateral acceleration of the nonlinear tyre model reaches a maximum but the linear model continues without curving, which is not realistic under these conditions.

3.3.4 Two Track Model with Linear Tyres

To make the two track model, the forces on each tyre was defined and used in the same force equalities and momentum equality as for the one track model, equations (3.7 - 3.9). Then the moment equalities around the x-axle and the y-axle was introduced as well as the force equality in the z-direction, equations (3.10 - 3.12). The centre of gravity is placed at the height h, see Figure 3.5.

$$ma_x = \sum_{i=1}^{4} F_{ix} \tag{3.7}$$

$$ma_y = \sum_{i=1}^{4} F_{iy} \tag{3.8}$$

$$J_z\ddot{\Psi} = f\sum_{i=1}^{2} F_{iy,front} + b\sum_{i=1}^{2} F_{iy,rear} \tag{3.9}$$

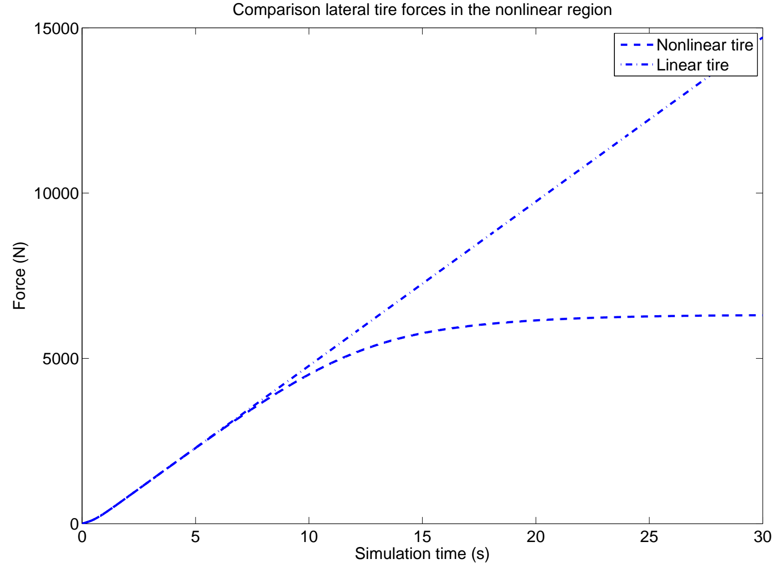


Figure 3.3: *Lateral tyre forces in the nonlinear region*

$$mg = \sum_{i=1}^{4} F_{iz} \tag{3.10}$$

$$ma_y h = \frac{t_w}{2} \sum_{i=1}^{4} F_{iz} \tag{3.11}$$

$$ma_x h = f \sum_{i=1}^{2} F_{iz,front} + b \sum_{i=1}^{2} F_{iz,rear} \tag{3.12}$$

In order to eliminate the roll angle (φ) from the equations it was assumed that the body is rigid, i.e. $\varphi_{front} = \varphi_{rear}$. Modelling the roll angle in front and in rear is done with the torsion stiffnesses in front (k_{front}) and in rear (k_{rear}) and then compared with the momentums ($\tau_{x,front}$ and $\tau_{x,rear}$) given by the tyre forces, see equations (3.13 - 3.16).

$$\tau_{x,front} = \frac{t_w}{2} F_{1z} + \frac{t_w}{2} F_{2z} \tag{3.13}$$

$$\tau_{x,rear} = \frac{t_w}{2} F_{3z} + \frac{t_w}{2} F_{4z} \tag{3.14}$$

$$\varphi_{front} = \frac{\tau_{x,front}}{k_{front}} \tag{3.15}$$

$$\varphi_{rear} = \frac{\tau_{x,rear}}{k_{rear}} \tag{3.16}$$

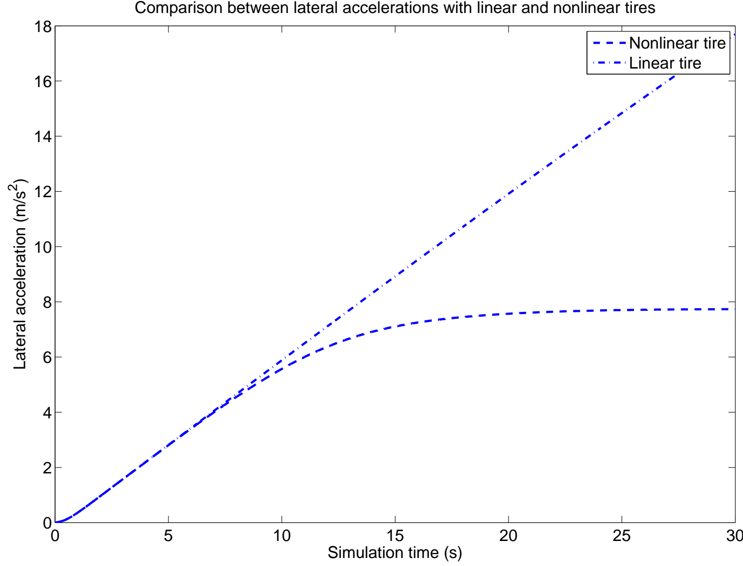


Figure 3.4: *Lateral acceleration in the nonlinear region*

3.3.5 Two Track with Roll and Load Distribution (Linear Tyres)

If the roll angle (φ) is assumed to be small it can be calculated accordingly

$$\varphi = \frac{ma_y h'}{c_{\varphi 1} + c_{\varphi 2} - mgh'} \tag{3.17}$$

where $c_{\varphi 1,2}$ are the axle roll stiffnesses front and rear [Pacejka, 2002]. The distance h' is the distance from the centre of gravity to the roll axle. The roll axle stretches from the height h_1 in front and the height h_2 in the rear. The load transfer ΔF_{zi} caused by the centripetal acceleration a_y is given by the equations

$$\Delta F_{z1} = \frac{1}{t_w}\left(\frac{c_{\varphi 1}}{c_{\varphi 1} + c_{\varphi 2} - mgh'}h' + \frac{b}{f+b}h_1\right) ma_y \tag{3.18}$$

$$\Delta F_{z2} = \frac{1}{t_w}\left(\frac{c_{\varphi 2}}{c_{\varphi 1} + c_{\varphi 2} - mgh'}h' + \frac{f}{f+b}h_2\right) ma_y \tag{3.19}$$

The longitudinally effects of weight distribution, acceleration and decelerating are taken into account by summarising the z-forces per tyre. The total z-forces on tyre$_1$ (front left tyre) is given by equation (3.20).

$$F_{1z} = \frac{b}{2(f+b)}mg - \Delta F_{1z} - \frac{h}{2(f+b)}ma_x + \frac{J_\varphi \ddot{\varphi}}{\frac{t_w}{2}} \tag{3.20}$$

The roll angle dynamics are modelled as in equation (3.21) where D_φ is the roll damping.

$$J_\varphi \ddot{\varphi} + D_\varphi \dot{\varphi} + (c_{\varphi,1} + c_{\varphi,2})\varphi = h_{roll} ma_y \tag{3.21}$$

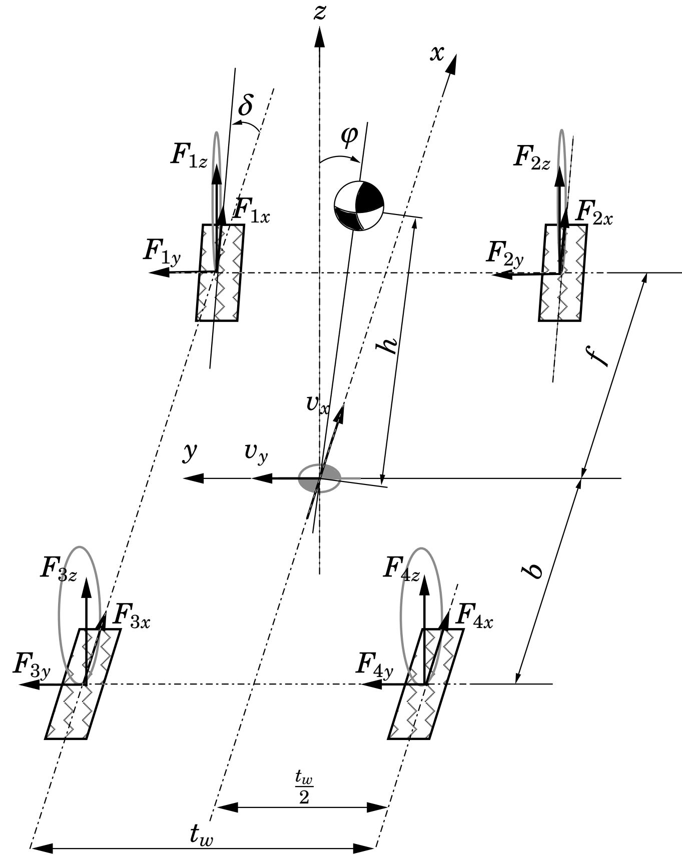


Figure 3.5: *Two track model.*

3.3.6 Two Track with Roll, Load Distribution and Magic Formula Tyres

The two track model that included roll and load distribution was then modified with non linear tyres. This was was done with the Magic Formula, see section 3.4.3.

3.4 Modelling Tyres

Modelling a tyre can be done in many different ways, with a wide span of complexity.

3.4.1 Slip Angle

A linear tyre model is a natural starting point. Linear tyres are approximated with a linear function. The angular difference between the direction in which a tyre is rolling and the wheel plane is called slip angle (α). When there is a difference between the directions a lateral force F_y arise. The

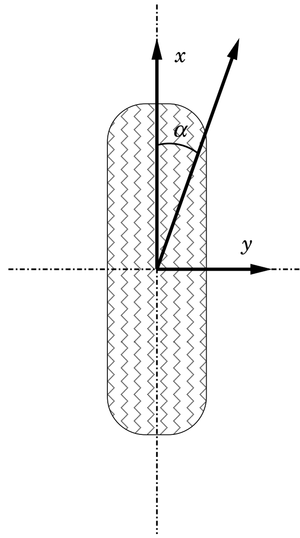


Figure 3.6: *Slip angle is defined as the difference between the wheel centre velocity and the wheel plane.*

definition of slip angle is

$$\tan(\alpha) = -\frac{V_y}{V_x}. \tag{3.22}$$

When the slip angel is small actually no sliding occurs, the difference between the two directions is due to the elasticity in the tyre. A tyre has the ability to roll in different direction compared to the direction defined by the wheel plane. For small angles the relationship between slip angles and cornering forces are linear but become nonlinear for large angles. This relation is called cornering stiffness and designates C_α see Figure 3.7 for a typical slip angel - lateral force relationship. As can be seen there are three different regions; elastic, transitional and frictional.

- **elastic** Linear relation between the lateral force F_y and slip angle α. The slip angle is a result of the elastic properties in the tyre and no sliding between the road and the tread occurs.

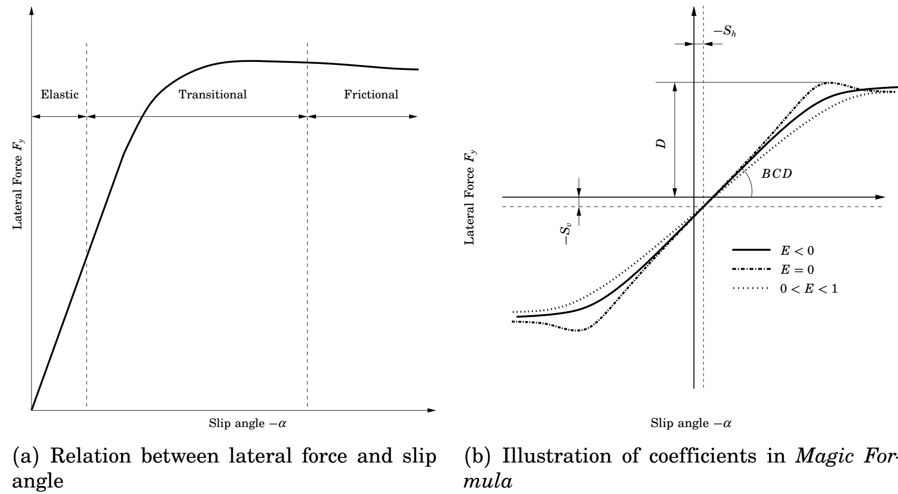
$$F_y = -C_\alpha \alpha \tag{3.23}$$

- **transitional** The relation is nonlinear and sliding occurs in a region of the thread.

- **frictional** The lateral force is starting to decrease, sliding occurs over the whole tread area.

The definition of *cornering stiffness* is the derivative at zero slip of the *lateral force* with regards to the *slip angle* curve. The negative sign in equation (3.24) is due to that it is more natural with a positive stiffness than a negative one.

$$C_\alpha = -\left(\frac{\partial F_y}{\partial \alpha}\right)_{\alpha=0} \tag{3.24}$$



(a) Relation between lateral force and slip angle

(b) Illustration of coefficients in *Magic Formula*

Figure 3.7: *Characteristic of tyres.*

3.4.2 Linear Tyre Model

This simple tyre model approximates the cornering stiffness with a linear function i.e. C_α is constant. This model is also modelled for pure cornering and does not handle longitudinal forces such as braking and traction. The model consist of the following relation between the slip angle and the lateral force.

$$F_y = -C_\alpha \alpha$$

The only parameter to specify in this model is the cornering stiffens C_α

3.4.3 Magic Formula for lateral slip

The magic formula is an semi empirical model invented by Bakker, Lidner and Pacejka. The formula fits measurements from a real tyre to a curve [Wennerström et al., 2005].

$$\begin{aligned} y(x) &= D\sin(C\arctan(Bx - E(Bx - \arctan(Bx)))) \\ Y(X) &= y(x) + S_v \\ x &= X + S_h \end{aligned} \tag{3.25}$$

The tyre formula contain six parameters

- B stiffness factor
- C shape factor
- D peak factor
- E curvature factor
- S_h horizontal shift
- S_v vertical shift

Where $Y(X)$ can be *lateral force*, *longitudinal force* or *self aligning torque*.

The influence that the parameters have on the tyre curve can be seen in Figure 3.7. This figure shows a *side force–slip angle* relation, the *Magic Formula* can also be used to describe the traction force and the self aligning torque of a tyre as well.

As can be seen, B affects the slope of the curve at $\alpha = 0$ hence the name *stiffness factor*. The maximum side force that could be attained is the value of D, the *peak factor*. By changing the parameter C the formula can be fitted to also describe brake force and self aligning torque. E shape the curvature for the horizontal position of the peak. S_h and S_v allows offsets to the curve with respect to the origin due to ply steer, rolling resistance, conicity and camber.

3.4.4 Longitudinal Slip

When traction and braking forces affect the tyre *longitudinal slip* occurs. This phenomena is due to the velocity difference in the thread of the tyre. Longitudinal slip denotes with κ and the definition is

$$\kappa = -\frac{v_x - r_e\omega}{v_x} \tag{3.26}$$

where

$$r_e = \frac{v_x}{\omega_0} \tag{3.27}$$

is the *effective rolling radius* and v_x is the wheel centre velocity. The effective rolling radius is the ratio between distance v_x that the wheel has travelled and the corresponding rotation angle speed ω_0. A wheel has a r_e less than the real radius r and larger than the loaded radius r_l, this due to compression of the tyre, see Figure 3.8. The minus sign in equation (3.26) is introduced

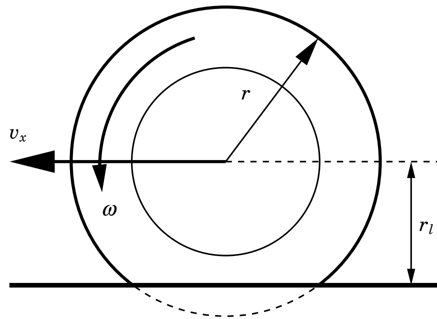


Figure 3.8: *Effective rolling radius $r_l < r_e < r$. In this picture the difference is highly exaggerating.*

so that positive κ means a positive longitudinal force. For a spinning wheel $r_e\omega$ is larger than v_x which results in a positive κ. For surfaces with low friction like ice and wet roads κ could be very large. The opposite situation with a breaking wheel gives a negative κ and when the wheel is locked the longitudinal slip becomes minus one. It is then natural that a free rolling wheel has zero slip.

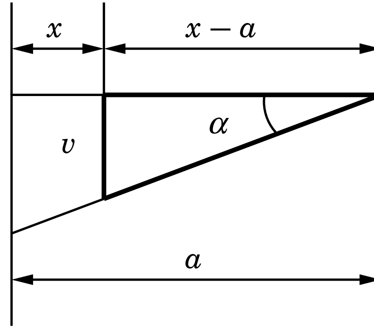


Figure 3.9: *Extract from Figure 3.10 showing how to calculate lateral displacement for a bristle.*

3.4.5 Magic Formula for Combined Slip

The magic formula can also be fitted to measurements from longitudinal forces and self aligning torque.

3.4.6 Brush Model

The brush model is an analytical model that describes lateral an longitudinal forces for a tyre. The idea is to model the rubber as a brush with small flexible bristles. The bristles are extended over the whole contact area in y direction but is indefinitely small in x direction.

Lateral Force and Aligning Moment

When slip occurs the bristles starts to deform. As depicted in Figure 3.10, the lateral displacement (v_α) in the adhesion region

$$v_\alpha = (x - a)\tan(\alpha) \tag{3.28}$$

see Figure 3.9 for how to obtain the above relation. The force that a single bristle gives arise to is $c_p v_\alpha$, this is a linear relation with the introduced lateral stiffness factor (c_p). The total force over the whole region is F_y and can be calculated with an integral as.

$$F_y = \int_{-a}^{a} c_p(x - a)\tan(\alpha) \tag{3.29}$$

Solving the integrand gives an expression for the lateral force.

$$F_y = -2a^2 c_p \tan(\alpha)$$

The aligning force is calculated in the same manner. The lateral force times the distance x and then integrate to get the total moment.

$$M_z = \int_{-a}^{a} x c_p(x - a)\tan(\alpha) = \frac{2}{3} c_p a^3 \tan(\alpha) \tag{3.30}$$

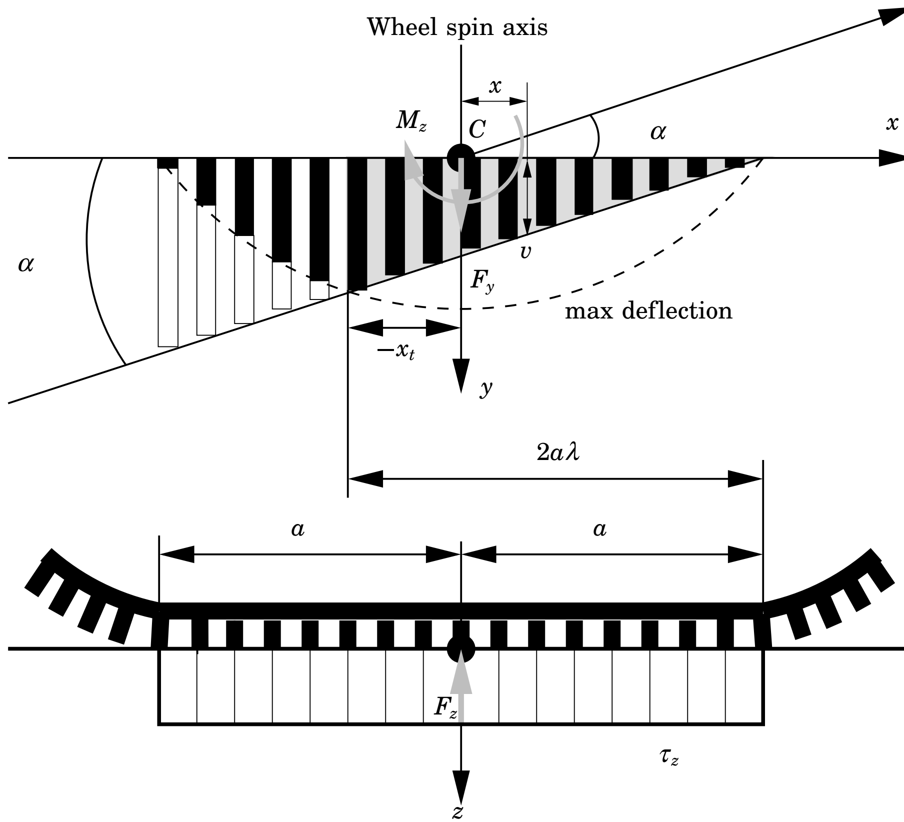


Figure 3.10: *The brush model for lateral forces(Upper top view and lower side view).*

Large Slip Angles

The above equations are only valid for small slip angles when the relation between force and slip angle is linear in the entire contact area. When the slip angle becomes large sliding occurs in one part of the tyre contact area. To model this, a parameter λ is added to the model. In the upper part of Figure 3.10 the shaded area bounded by $2a\lambda$ and the lateral displacement v, is the adhesion region. In this part the linear relation is still valid. The other part is a sliding region where a nonlinear force relation comes up.

The pressure that F_z gives arise to is equal distributed over the contact area $2a$

$$\tau_z = \frac{F_z}{2a} \tag{3.31}$$

The lateral force for a bristle is limited by the above pressure and the road tyre friction.

$$|c_p(x-a)\tan(\alpha)| \leq \mu\tau_z \tag{3.32}$$

Sliding must then occur when the force becomes equal or larger than $\mu\tau_z$.

This *sliding point* denotes with x_s

$$-c_p \tan(\alpha)(x_s - a) = \mu\tau_z$$

$$\text{solving for } x_s \text{ gives} \tag{3.33}$$

$$x_s = -\frac{\mu\tau_z}{c_p \tan(\alpha)} + a$$

The parameter λ describes how large the adhesion-sliding part is relation to the entire contact area.

$$\lambda = \frac{a - x_s}{2a} \tag{3.34}$$

The side force has to be split into two parts, one from the sliding ($-a \to a(1-2\lambda)$) and one from the adhesion region ($a(1-2\lambda) \to a$). The total force from the two regions becomes

$$F_y = -\int_{-a}^{a(1-2\lambda)} \frac{\mu F_z}{2a} dx + \int_{a(1-2\lambda)}^{a} c_p \tan(\alpha)(x - a) dx \tag{3.35}$$

and solved

$$F_y = -\underbrace{\mu F_z}_{2C_\alpha \tan(\alpha)\lambda}(1-\lambda) - \lambda^2 \underbrace{2a^2 c_p}_{C_\alpha} \tan(\alpha) \tag{3.36}$$

The rewritings above comes from

$$\lambda = \frac{a - x_s}{2a} = \frac{\mu\tau_z}{2ac_p \tan(\alpha)} = \frac{\mu F_z}{4a^2 \tan(\alpha)} = \frac{\mu F_z}{2C_\alpha \tan(\alpha)} \tag{3.37}$$

where equation (3.33) and (3.31) have been used. After further simplifications the force can be written as

$$F_y = -2C_\alpha \tan(\alpha)\lambda(1+\lambda) - C_\alpha \tan(\alpha)\lambda^2 =$$
$$C_\alpha \tan(\alpha) \underbrace{(\lambda(2-\lambda))}_{f(\lambda)} \tag{3.38}$$

where

$$f(\lambda) = \begin{cases} \lambda(2-\lambda) & \lambda \leq 1 \\ 1 & \lambda < 1 \end{cases} \tag{3.39}$$

Slip starts at the longitudinal position given by

$$\lambda = \frac{F_z \mu}{2C_\alpha |\tan(\alpha)|} \tag{3.40}$$

The relation for aligning moment also differ with large slip angles. Similar results as in the case of lateral slip is obtained, the same method is used.

The contact area is split into two pieces, one adhesion- and one slip area. The λ parameter describes where the border between those areas is located. The total moment is split up and described by two separate contributing parts.

$$M_z = -\int_{-a}^{a(1-2\lambda)} x\frac{\mu F_z}{2a} dx + \int_{a(1-2\lambda)}^{a} x c_p \tan(\alpha)(x - a) dx \tag{3.41}$$

$$M_z = -\mu F_z a\lambda(\lambda - 1) + \frac{a^3 c_p \tan(\alpha)}{6}(-12\lambda^2 + 16\lambda^3) \tag{3.42}$$

or

$$M_z = C_m \tan(\alpha) 6\lambda^2(1-\lambda) + C_m \tan(\alpha)\lambda^2(4\lambda - 3) \tag{3.43}$$

with

$$C_m = C_\alpha \frac{a}{3} = \frac{2a^3 c_p}{3} \quad \text{and} \quad \mu F_z = 2C_\alpha \tan(\alpha)\lambda \tag{3.44}$$

Expression for C_α is found in equation (3.37). The sum results in an expression for the aligning torque.

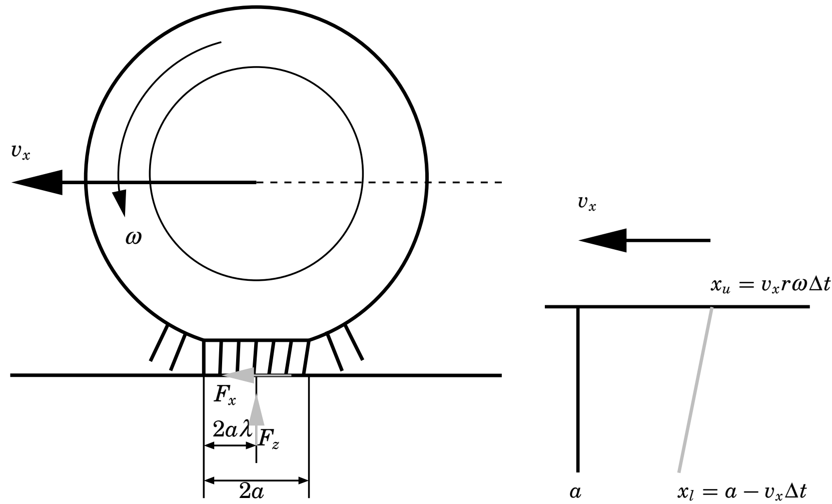
$$M_z = C_m \tan(\alpha) g(x) \tag{3.45}$$

$$g(\lambda) = \begin{cases} \lambda^2(3-2\lambda) & \lambda \leq 1 \\ 1 & \lambda < 1 \end{cases} \tag{3.46}$$

$$\lambda = \frac{F_z \mu}{2C_\alpha |\tan(\alpha)|} = \frac{a F_z \mu}{6C_M |\tan(\alpha)|} \tag{3.47}$$

Longitudinal Slip

If there is a difference between the angular wheel velocity and the free rolling velocity, slip arises as described above. If this is the case the slip deform the bristles as in Figure 3.11. Direction of the bristles deformation depends on the type of moment acting on the wheel. If it is a braking force the velocity is slower in the upper part of the bristles relative to the lower part, for traction it is the opposite situation. The difference or the longitudinal displacement



(a) Difference between angular velocity ω and free rolling wheel gives arise to a longitudinal slip. A traction moment is acting on the wheel.

(b) Bristle deformation during traction

Figure 3.11: *Longitudinal slip for the brush model.*

is thus

$$v_\kappa = x_l - x_u = r\omega\Delta t - v_x\Delta t = \frac{r\omega - v_x}{v_x}(v_x\Delta t) \tag{3.48}$$

This can be written with the definition of longitudinal slip equation (3.26).

$$\kappa(a - x) \tag{3.49}$$

This is very similar to the lateral slip displacement with the difference that $\tan(\alpha)$ is substituted with κ.

Once again the total force over the contact region be written in the same manner as for lateral slip, by introducing a stiffness C_κ and integrate, this results in

$$F_x = C_\kappa \kappa h(x) \tag{3.50}$$

$$h(\lambda) = \begin{cases} \lambda(2-\lambda) & \lambda \leq 1 \\ 1 & \lambda < 1 \end{cases} \tag{3.51}$$

with

$$\lambda = \frac{F_z\mu}{2C_\kappa|\kappa|} \tag{3.52}$$

Combined Lateral and Longitudinal Forces

The idea is now to combine the above described slips and write a simple combined relation, the combined slip can be written as

$$\begin{aligned} \sigma_x &= C_\kappa \kappa \\ \sigma_y &= -C_\alpha \tan(\alpha) \\ \sigma &= \sqrt{\sigma_x^2 + \sigma_y^2} \end{aligned} \tag{3.53}$$

This results in a complete tyre model that consider slip in both lateral and longitudinal directions.

$$\begin{aligned} F_x &= C_\kappa \kappa f(\lambda) \\ F_y &= -C_\alpha \tan(\alpha) f(\lambda) \\ f(\lambda) &= \begin{cases} \lambda(2-\lambda) & \lambda \leq 1 \\ 1 & \lambda < 1 \end{cases} \\ \lambda &= \frac{F_z\mu}{2\sigma} \end{aligned} \tag{3.54}$$

4

Optimisation

With the spline package implemented in DYMOLA, the next step was to optimise simple vehicle models. In order to get a feel for the optimisation tools, tests on even simpler mechanical systems were performed. Below are some simple examples that show that it is not always the correct parameter value that is received although the AMPL solver indicates that, problem with local minimum can occur.

4.1 The Optimica Compiler

The Optimica Compiler (TOC) is a tool that was used to optimise parameters in vehicle models. TOC is written by Johan Åkesson, PhD student at the Department of Automatic Control, Lund University, Lund, Sweden. During this master thesis, TOC was under construction and was updated repeatedly.

An optimisation problem to be solved with help of TOC consists of a model for the system written in Modelica, an Optimica file containing the description of the problem and an optional initial guess. TOC then combines the files and generate a new set of files written in AMPL code. This code is then to be executed in AMPL to obtain a solution to the problem, see Figure 4.1. For a more detailed explanation and how to install TOC, see [Åkesson Johan, 2007].

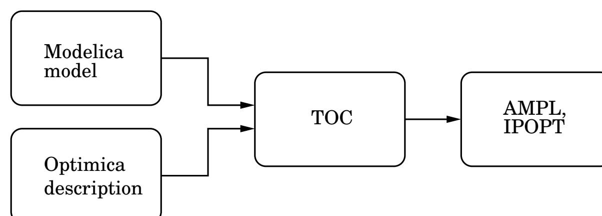


Figure 4.1: *Flowchart over the steps involved in running an optimisation problem*

4.2 Dynamic Optimisation

The optimisation method used in this thesis is a simultaneous method, or a *direct* transcription. This approach is a fairly new branch, it has existed for about twenty years. Older approaches are the *indirect* methods dynamic programming and the maximum principle. Direct methods fully discretise the states and control variables which leads to large-scale, but sparse, non linear programming (NLP) problems, see section 4.2.1. These methods couple directly to the solution of the differential-algebraic equations (DAE) and is solved once, at the optimal point. The methods have the ability to handle large scale problems and it is possible to put path constraints on states and controls. The orthogonal collocation is algebraically stable and converges for unstable systems, it is also equivalent to the implicit Runge-Kutta [Biegler et al., 2001].

4.2.1 Orthogonal Collocation

One way to discretisise differential equations is orthogonal collocation. The idea of this approach is to split the function into a finite set of points $a = t_1 < \ldots < t_n = b$, these points t_n are called *collocation points*. The task is to fit an analytic solution consisting of a linear combination of basic functions to the differential equation according to equation (4.1). The approximation must satisfy the differential equation in every collocation point [Heath, 2002].

$$u(t) \approx v(t, \mathbf{x}) = \sum_{i=1}^{n} x_i \varphi_i(t) \tag{4.1}$$

The basis functions can for example be polynomials, B-splines or trigonometric functions. In TOC Langrange polynomials are used as basic functions $\varphi_i(t) = L_i(t)$.

The Lagrange polynomials are defined as

$$L_i(t) = \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{(t - t_k)}{(t - t_k)} \qquad i = 1, \ldots, n \tag{4.2}$$

$$L_j(t_i) = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases} \qquad i, j = 1, \ldots, n \tag{4.3}$$

4.3 AMPL and IPOPT

AMPL is a modelling language for mathematical programming. It is a high-level programming language for describing and solving large scale optimisation and scheduling problems. AMPL does not solve these problems directly, this is done by external solvers, for example IPOPT, CPLEX and KNITRO. The solver used in this thesis is the IPOPT. AMPL handles both linear and non-linear problems. AMPL was developed at Bell laboratories. For more information visit AMPL on the web [AMPL®, 2007].

IPOPT stands for Interior Point OPTimizer and is an open software package for large scale nonlinear optimisation. It is designed to find solutions of mathematical optimisation problems in the form of

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f(x) \\ &\text{subject to} \quad g^L \leq g(x) \leq g^U \\ &\qquad\qquad\quad x^L \leq x \leq x^U \end{aligned}$$

where $x \in \mathbb{R}^n$ are the optimisation variables with possible lower and upper bounds, x^L and x^U. The constraints, $g(x)$ have upper and lower bounds g^L and g^U. The functions $f(x)$ and $g(x)$ can be both linear and nonlinear [IPOPT, 2007].

4.4 Intermediate Spline Solutions

During the progress of this thesis there was initially no support for the newly constructed spline package in TOC and AMPL. A first temporary solution was to create splines in MATLAB and import these into DYMOLA, see section 4.4.1. Later it became possible to run **Splines1.1** with a solution that reflects the future function in TOC, it is described in section 4.4.2.

4.4.1 Matlab Splines

The first temporary solution was a MATLAB script that construct "splines" from the DYMOLA generated result file. These splines were if-statements that returned a value depending on in which interval the evaluation point was placed. The work flow is depicted in Figure 4.2 and can be described

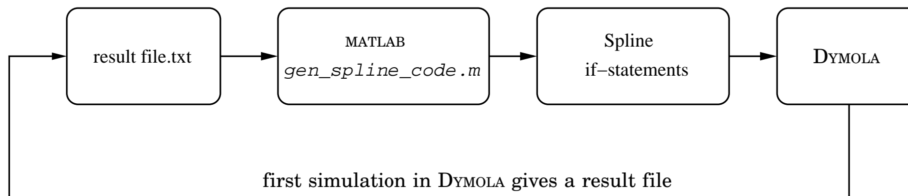


Figure 4.2: *Splines were created with a MATLAB script.*

as:

1. simulate *real* process
2. run `gen_spline_code.m`, which produces Modelica files containing if--statements and the spline polynomial expressions based on the simulation result file.
3. simulate splines and estimation model. This result in a initial guess.
4. run TOC which generates the AMPL files
5. use AMPL/IPOPT to solve the optimisation problem

4.4.2 AMPL Splines

Later in the thesis work it was made possible to run the spline package **Splines1.1** in AMPL via an AMPL function. The AMPL function is the same function that will be supported in later versions of TOC. The work flow can be seen in Figure 4.3 and the steps are as follows:

1. simulate *real* process
2. run `gen_incs.m`, which produces a C header file with the data from the real model
3. include the header file in the AMPL `funcadd.c` to make the data available and write a function within `funcadd.c` that uses the desired functions from the spline package **Splines1.1**.
4. declare the function in the AMPL `<file>.run`
5. use the function in AMPL cost function, `<file>.Cost.mod`

Since **Splines1.1** uses single precision it is necessary to increase the tolerance in AMPL to $1e^{-5}$, otherwise it is possible that IPOPT does not find a point that fulfills the optimality condition at the tolerance specified. Example files to run an AMPL spline can be seen in Appendix C.

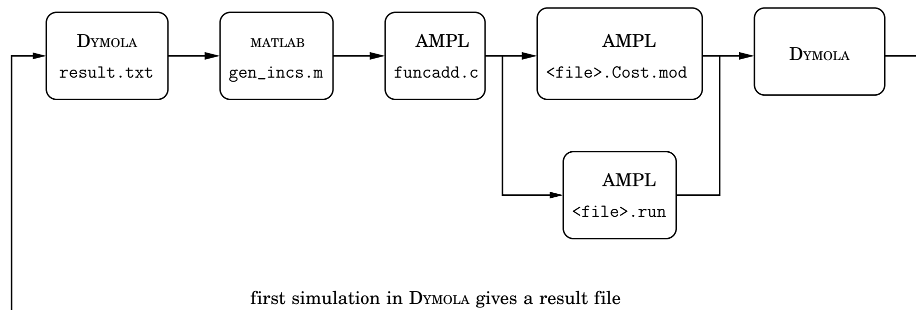


Figure 4.3: *Steps involved to run an AMPL spline*

4.5 Parameter Optimisation

A parameter estimation problem can be treated as an optimisation problem. For a parameter optimisation problem there has to be two systems, one process treated as the real and one model containing the unknown parameters that are to be estimated. The task is to compare these two systems and minimise the deviations, see Figure 4.4. Below a cost function that integrates the error between the process states $\mathbf{y_p}$ and the model states $\mathbf{y_m}$ is shown. The optimal solution is found when this function is minimal.

$$\min \int_0^t (y_p - y_m)^T Q (y_p - y_m) dt$$

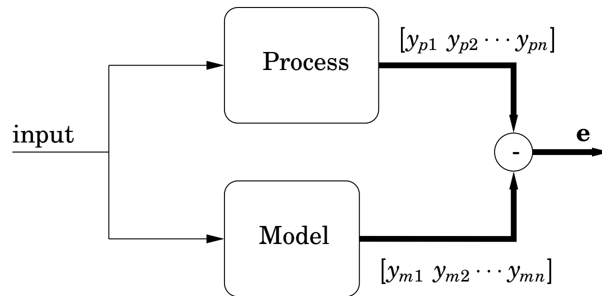


Figure 4.4: *Process and model are excited with the same input and the difference in output will be minimised.*

The cost function could consist of more than just one state, if this is the case it is necessary with weights (Q) on the different model states due to different sizes of the states.

4.5.1 Servo Optimisation

A simple model for parameter optimisation consisting of two masses combined with a spring and a damper will be used to illustrate some common phenomena. The aim is to find the best value of one or more unknown parameters, eg. the spring constant, damper constant or one of the two masses. It is also possible to estimate more than just one parameter at time.

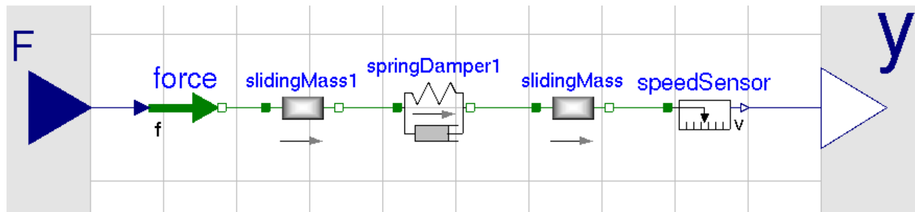


Figure 4.5: *The model structure in* DYMOLA.

One has to be careful when several parameters are to be estimated, erroneous combinations of parameters could be found and the optimisation tool would find a local minimum as the optimal solution. This scenario does not just happen when identification of several parameters are involved, it could also happen when initial guesses are too far away from the correct values or too large parameter intervals are used in the Optimica constraints.

Another aspect to keep in mind is the choice of input to the two systems. To get a good estimation of parameters it is important to excite the system with a good enough input signal. In the case with an unknown mass in the servo described above, the input to the systems was a sinusoid.

The system with a unknown parameter m_1 has the initial guess $m_1 = 0.7$ the real servo has $m_1 = 1$. Outputs of the two systems can be seen in Figure 4.6. After a run in the optimisation tool, an estimate of the mass is $m_1 = 0.977807$. The major reason for the difference is the accuracy in the splines that were used to represent the original process data. As could be

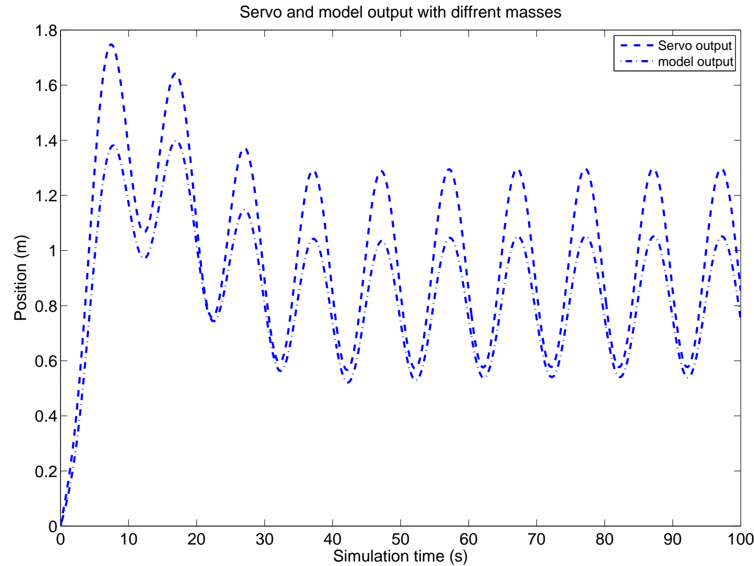


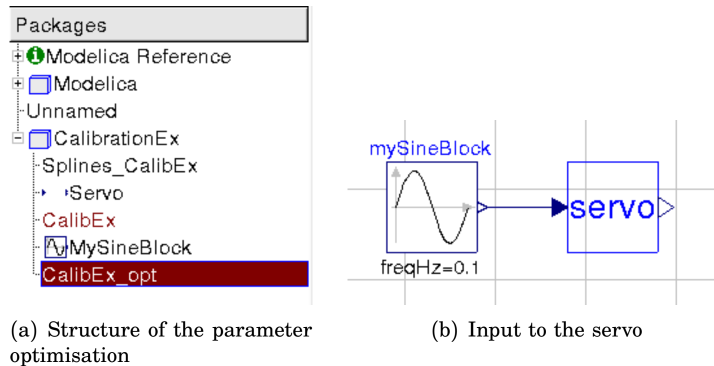
Figure 4.6: *A comparison of the two servo systems, which have the same model structure, show different outputs due to different masses.*

seen in the `optModel` class below the allowed interval for m_1 ranges from 0.3 to 3. This interval can in this case be stretched much further and still a correct optimal solution will be found. To provoke the optimisation problem, the initial guess is changed to $m_1 = 0.05$ but still the same optimal solution is found. This due to that it is a rather simple example with low complexity.

```
class optModel
  /*Which parameter that will be optimised*/
  oq servo.m1(lowerBound=0.3,upperBound=3);
optimization
  grid(finalTime=fixedFinalTime(finalTime=100),
       nbrElements=100);
 /*Cost-function*/
  minimize(lagrangeIntegrand=(servo.y-sp.servo_y)^2);
subject to

end optModel;
```

To make the problem more complex and to find some limitations, two parameters can be optimised simultaneously. With narrow intervals for the parameters the estimation gives excellent parameters $m_1 = 0.999969$ and $c_1 = 0.0101851$. A wider parameter interval result in an other *optimal* solution $m_1 = 5.58$ and $c_1 = 0.731468$. In general, optimisation of several parameters at the same time gave a more precise solution than one parameter optimisatiom at a time, as long as the parameters do not depend too much on each other.



(a) Structure of the parameter optimisation

(b) Input to the servo

Figure 4.7: *The dynamics is modelled in Modelica and the optimisation specifications are written in Optimica code*

```
oq servo.m1(lowerBound=0.1,upperBound=10);
oq servo.c1(lowerBound=0.001,upperBound=4);
```

This clearly shows that one has to be aware of the difficulty of setting up the optimisation problem, to keep an eye on the total cost function and to carefully evaluate obtained solutions. Solutions could be found in a local minimum, different from the global minimum. For example, the cost functions differs in the above simulations by a factor of 66 000.

- Objective for first optimisation $8.63e^{-4}$
- Objective for second optimisation 57.33

Outputs from the two optimised systems are in Figure 4.8. Some physical parameters are however easy to measure or manually estimate, examples are masses and lengths. Here, a mass of $5.5kg$ should be considered unreasonble. It is advisable to have a clue about the size of the estimated parameter. Without this knowledge, approximate values of the parameters may be found by means of simple experiments. Those values may then serve as initial guesses to the optimisation procedure and to get the last fitting, optimisation can be used.

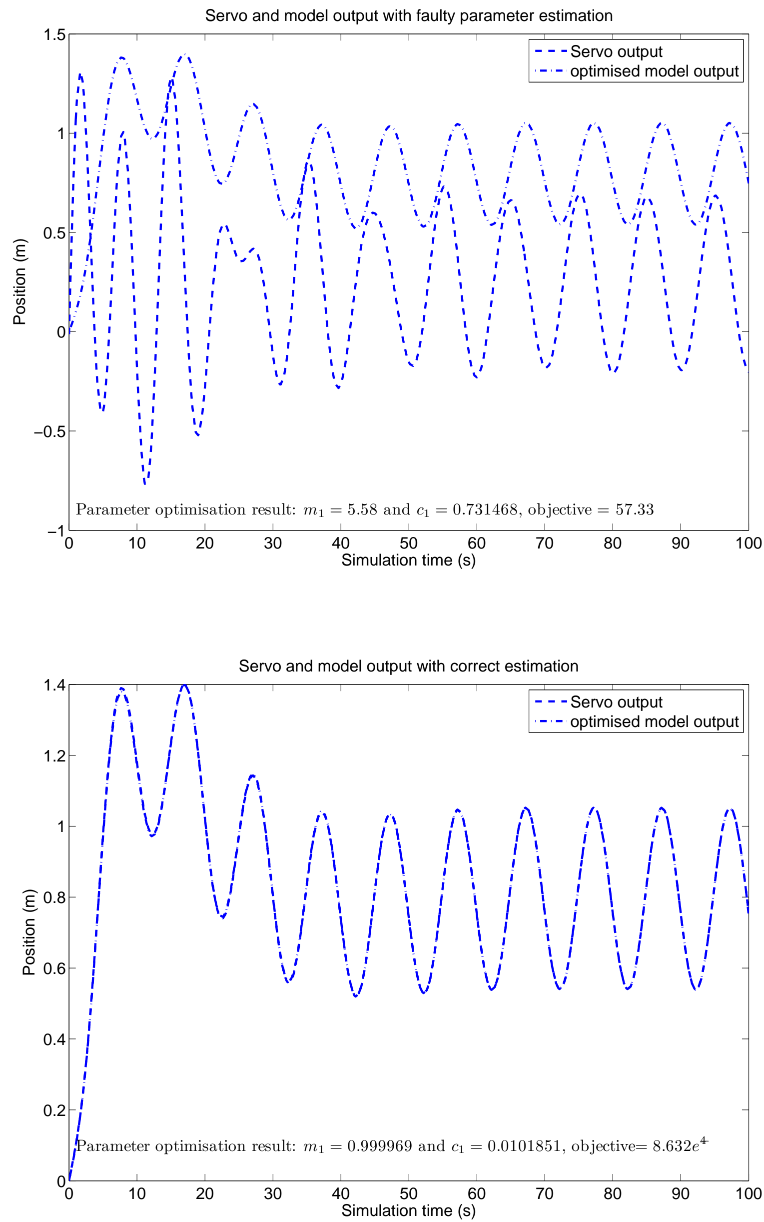


Figure 4.8: *Two* optimal *solutions for the spring-mass system. Upper optimisation results in a* local *minimum due to wide parameter interval. Lower figure shows the correct parameter optimisation, in this case a narrow parameter interval has been used, and the correct* global *minimum is found by the optimisation algorithm.*

5

Parameter Optimisation in Vehicle Models

In this chapter results of experiments on simple vehicle models are presented. The starting point is a simple One Track model with a few degrees of freedom consisting of only two wheels. The models that were used during the experiments can bee found in Chapter 3

5.1 One Track Model

The first model created was the One Track model and it was used to get a feel for the optimisation process and evaluate how well parameters could be determined. The One Track model has two inputs; velocity (v_{in}) and steering wheel angle (δ). In reality, forces and velocities for a vehicle are difficult to measure with high accuracy, while accelerations are easier to obtain. Therefore it is the accelerations that are used in the cost functions.

5.1.1 One Track Model with Linear Tyres

The initial optimisation was to compare two One Track models to each other and see if good parameter values could be obtained.

Model One Track with linear tyres

Optimisation model One Track with linear tyres

Parameters to be optimised Inertia (J_z), Cornering stiffness front (C_{12}) and rear (C_{34})

Input Steering angle (δ) is a ramp from 0-2 rad
Longitudinal velocity (v_x) is constant at $15 m/s$

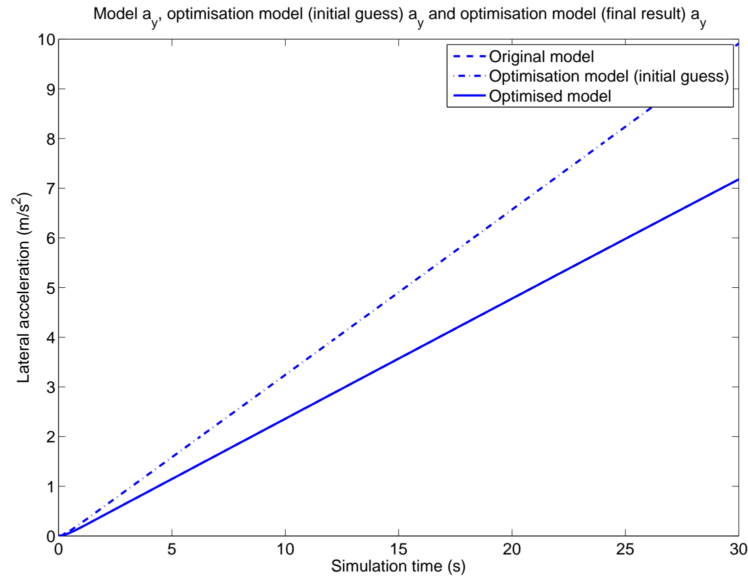


Figure 5.1: *Original, simulated initial guess and optimised a_y*

Vehicle states in cost function Lateral acceleration (a_y)

Simulation time $30s$

Optimisation time $<$ 1min

The cost function minimises the quadratic error, here the error in a_y, see Figure 5.1. The original data is represented by splines.

```
minimize(lagrangeIntegrand = (car.a_y-spline_a_y.car_a_y)^2);
```

The parameter values and the result from the optimisation can be seen in Table 5.1 and Figure 5.1. As can be seen, the optimised results compare well with the original values.

Parameter	Original	Init guess	TOC limits	Optimised	Cost
J_z	2800	5000	100-10000	2800.97	5.05e-8
C_{12}	40000	100000	1000-200000	40000.1	5.05e-8
C_{34}	50000	100000	1000-200000	49999.9	5.05e-8

Table 5.1: *Parameter optimisation results for One Track Model.*

5.1.2 One Track Model with Nonlinear Tyres

The next step was to extended the One Track with nonlinear tyres based on the *Magic Formula* [Pacejka, 2002]. The nonlinear model is considered

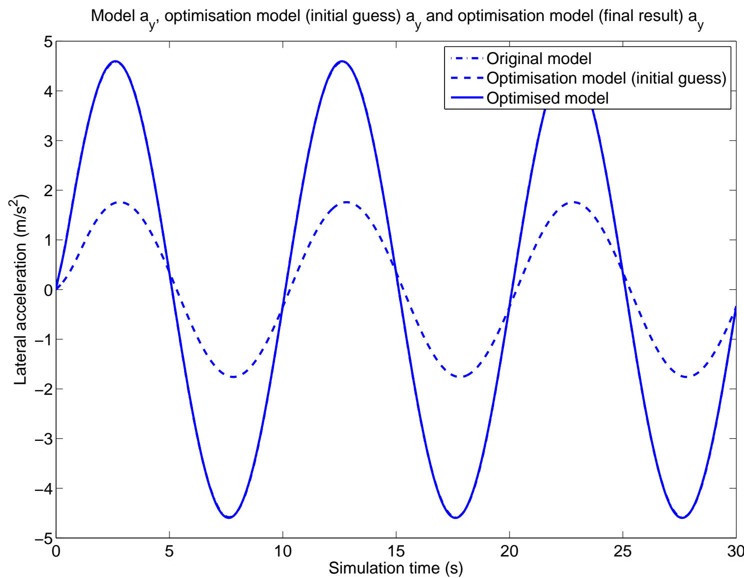


Figure 5.2: *Original, simulated initial guess and optimised a_y for linear model versus non-linear*

to be the reference model that the linear should be optimised to follow as close as possible. In this experiment there are non matching parameters between the two models and some parameters are unknown from the start. The nonlinear tyre model has four parameters and the linear model only one.

Model One Track with nonlinear tyres

Optimisation model One Track with linear tyres

Parameters to be optimised Cornering stiffness front (C_{12}) and rear (C_{34})

Input Steering angle (δ) is a sinus with amplitude 2 and frequency 0.1 Hz
Longitudinal velocity (v_x) is constant at $10m/s$

Vehicle states in cost function Lateral acceleration (a_y)

Simulation time $30s$

Optimisation time $<$ 1min

The results from the optimisation can be seen in Table 5.2 and Figure 5.2, and it shows that the cost function was acceptable and that the optimised a_y follows the original very well. Since there are no tyre parameters to compare with directly, and to demonstrate how the vehicle models behaves, the global positions have been plotted in Figure 5.3. The two trajectories are almost a perfect match, which implies that the tyre models are within the nonlinear region.

Parameter	Original	Init guess	TOC limits	Optimised	Cost
C_{12}	*Unknown*	70000	1000-200000	62030.8	4.05e-3
C_{34}	*Unknown*	60000	1000-200000	57987.6	4.05e-3

Table 5.2: *Parameter optimisation results for linear One Track Model versus nonlinear when the tyres are in the linear region.*

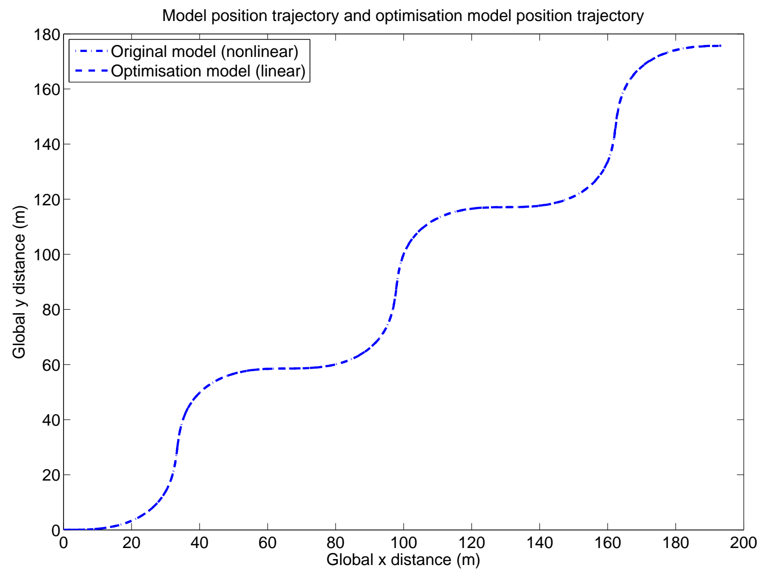


Figure 5.3: *Global model positions after optimisation, the tyres are in the linear region*

To evaluate what happens if the tyres operates in the nonlinear region, the experiment is redone with a larger velocity. The parameter optimisation results can be seen in Table 5.3 and the cost function indicates that it is not a good match. The different lateral accelerations involved can be seen in Figure 5.4. To further see the result, the global position is plotted in Figure 5.5 where it clearly shows that the linear model can not follow the nonlinear.

Model One Track with nonlinear tyres

Optimisation model One Track with linear tyres

Parameters to be optimised Cornering stiffness front (C_{12}) and rear (C_{34})

Input Steering angle (δ) is a sinus with amplitude 2 and frequency 0.1 Hz
Longitudinal velocity (v_x) is constant at $20m/s$

Vehicle states in cost function Lateral acceleration (a_y)

Simulation time $30s$

Optimisation time $<$ 1min

Parameter	Original	Init guess	TOC limits	Optimised	Cost
C_{12}	*Unknown*	10000	1000-200000	3390.18	1.63e+3
C_{34}	*Unknown*	20000	1000-200000	3158.05	1.63e+3

Table 5.3: *Parameter optimisation results for linear One Track Model versus nonlinear when the tyres are in the nonlinear region.*

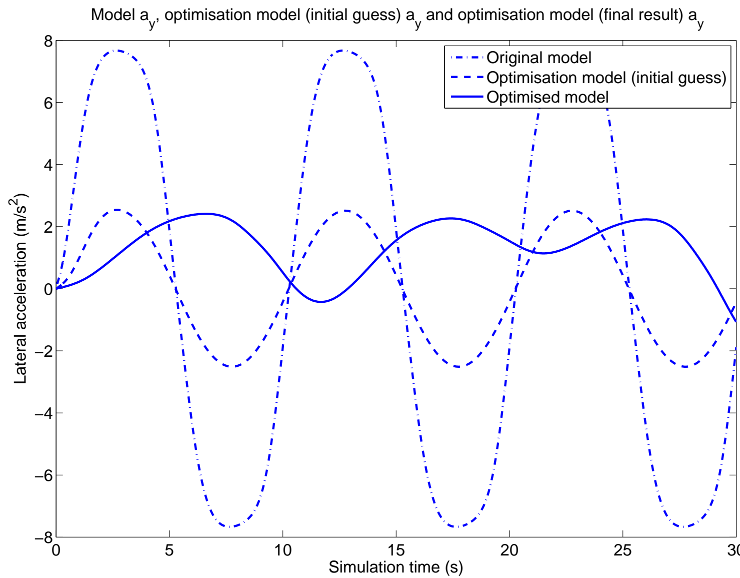


Figure 5.4: *Original, simulated initial guess and optimised a_y for linear model versus nonlinear in the nonlinear region*

5.1.3 Optimisation of Nonlinear Tyre Parameters

The accuracy of an optimisation of the nonlinear parameters in the tyre model (Magic Formula [Pacejka, 2002]) is evaluated in this experiment. Two One Track vehicle models, both with nonlinear tyres are used and one is to be optimised to follow the other. The parameter values and results can be seen in table 5.4. All the parameter values was found with high accuracy. The states used in the cost function can be seen in Figures 5.6-5.7 and it can be seen that the optimised model follows the original very well.

Model One Track with nonlinear tyres

Optimisation model One Track with nonlinear tyres

Parameters to be optimised The Magic Formula parameters for the front tyre: B_{12}, C_{12}, D_{12}, E_{12}

Input Steering angle (δ) is a ramp from 0-2 rad
Longitudinal velocity (v_x) is ramp from 0-20 m/s

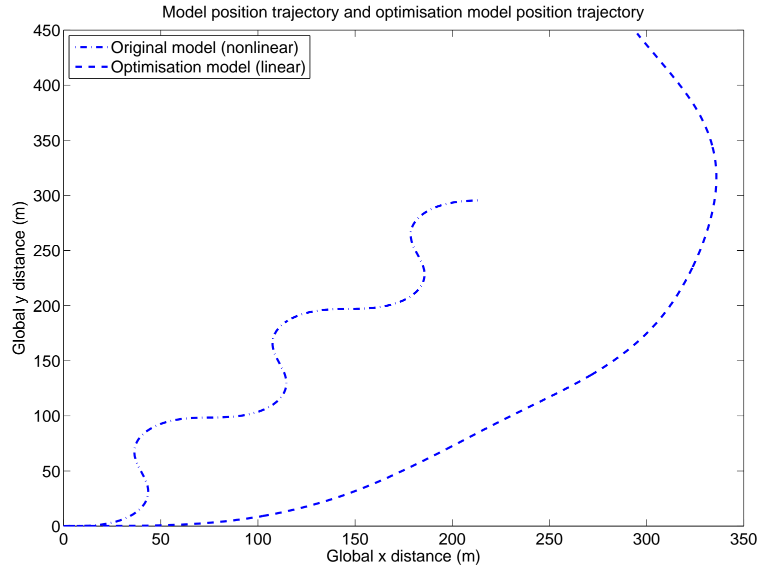


Figure 5.5: *Global model positions for linear model versus nonlinear in the nonlinear region after optimisation*

Vehicle states in cost function Lateral acceleration (a_y) and Yaw acceleration ($\ddot{\Psi}$)

Simulation time $30s$

Optimisation time 1min

Parameter	Original	Init guess	TOC limits	Optimised	Cost
B_{12}	7.69231	2	1-10	7.67933	2.1017e-4
C_{12}	1.3	1	0.5-2	1.30299	2.1017e-4
D_{12}	6307.2	4000	1000-10000	6305.63	2.1017e-4
E_{12}	-2	-1	-3-(-0.5)	-1.99374	2.1017e-4

Table 5.4: *Parameters in magic formula optimised.*

5.2 Optimisation and VDL

The most advanced parameter optimisation that was performed was the VDL optimisation.

5.2.1 Vehicle Manoeuvres

With more complex models two standard manoeuvres were also tried.

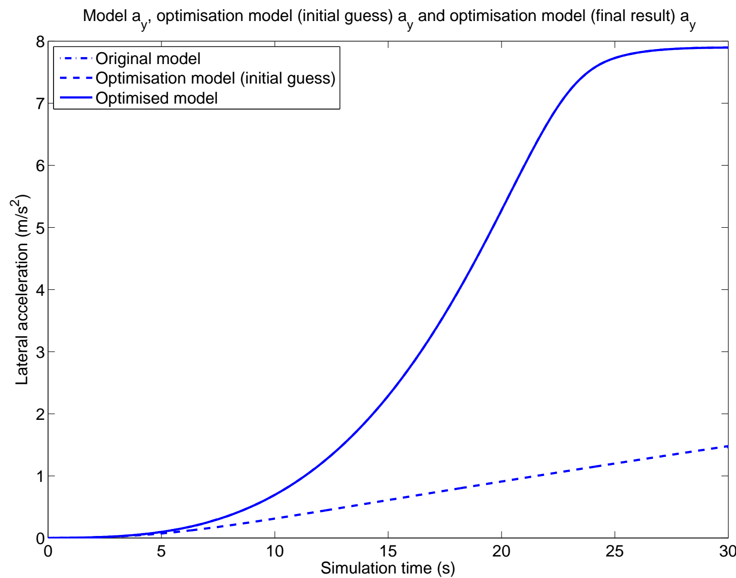


Figure 5.6: *Original, simulated initial guess and optimised a_y for optimisation of Magic Formula parameters*

Fishhook

A fishhook manoeuvre starts with a substantial turn in one direction and then another in the opposite direction. The trajectory looks like a fishhook, hence the name. This manoeuvre is usually used by the automotive industry for testing rollover behaviour.

Lane Change

The lane change manoeuvre can be described by a single period sinus. A double lane change consists of two consecutive, single period, sinuses with different signs.

5.2.2 Parameter Optimisation Procedure

With models from Vehicle Dynamics Library (VDL) generating reference data, the optimisation has to be divided into *sub-optimisations*. This because static and dynamic parameters can not be estimated at the same time with acceptable results. To optimise a parameter separately, the inputs to the model have to be chosen carefully, ie. run a specific experiment. Features for the models are stated in Section 3.1. Below a summary of models and possible inputs to estimate certain parameters.

1. **Estimate steady state parameters**

   Model choice One track, Two track

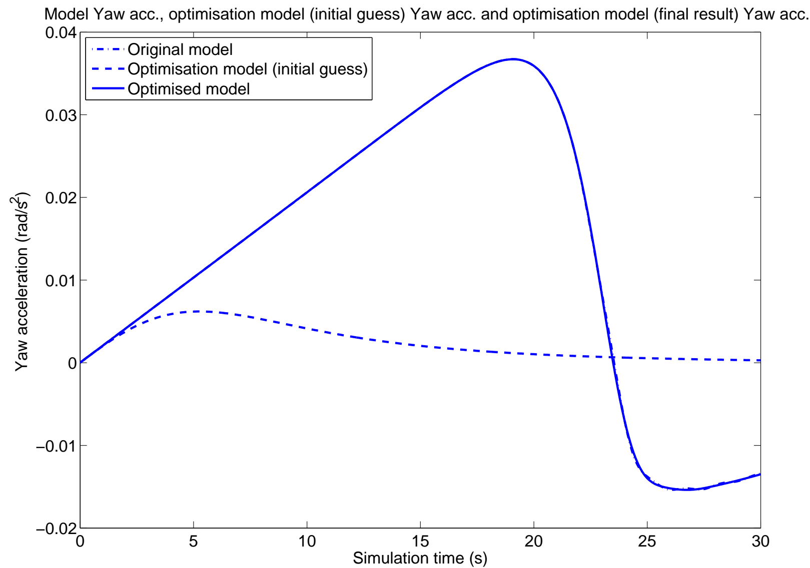


Figure 5.7: *Original, simulated initial guess and optimised* $\ddot{\Psi}$ *for optimisation of Magic Formula parameters*

Input choice Constant force/torque in longitudinal direction and ramp for steering angle

Vehicle states in cost function lateral acceleration, yaw rate

2. **Estimate dynamic parameters**

Model choice Two track with roll and pitch

input choice Constant force in longitudinal direction and eg. *lane change* for roll, alternating force in longitudinal direction and constant steering angle for pitch or a *fishhook*

Vehicle states in cost function Roll rate, yaw acceleration

5.2.3 VDL model

This optimisation used data generated from a model from VDL. The model is the *CompactLKRill* chassi with a McPherson suspension in front and a trailing arm in the rear. It is a large difference in complexity between the VDL model and the optimisation model. To get acceptable results from the estimation, the above described optimisation procedure was intended to be used. Tries with *all parameter* estimation were also performed without an acceptable result. The first step is to optimise the steady state parameters. This is done with the simple Two Track with linear tyres.

Model VDL model

Optimisation model Two Track with linear tyres

Parameters to be optimised Cornering stiffness front (C_{12}) and rear (C_{34}), vehicle mass (m), distance from centre of gravity to rear axle (b) and height of centre of gravity (h)

Input Ramp as steering angle and constant force in longitudinal direction

Vehicle states in cost function Lateral acceleration (a_y) and yaw rate ($\dot{\Psi}$)

Simulation time 10s

Optimisation time 20 min

Results from the experiment are presented in Table 5.5, there can it be seen that the cornering stiffnesses are to large. The original states, the initial guesses and the resulting states in the cost function can be seen in Figures 5.8-5.9.

Different inputs to the systems and different states in the cost function were tried. In Table 5.6 are parameters for a run with a lane change manoeuvre. This resulted in a better result for cornering stiffness. But mass and centre of gravity height are not close to the original. This however, do not necessarily mean that it is unrealistic parameters. There are lots of dynamics in the VDL model that the reduced model do not handle, compromises between parameters must take place. The original states, the initial guesses and the resulting states in the cost function can be seen in Figures 5.10-5.11.

With more time to spend on these optimisation issues it would most likely result in better parameter values.

Parameter	Original	Init guess	TOC limits	Optimised	Cost
C_{12}	72000	70000	30000-800000	280902	5.125e-3
C_{34}	72000	60000	30000-800000	170188	5.125e-3
h	0.48	0.5	0.1-2	1.05012	5.125e-3
b	2.24	2.2	0.5-2.47	1.91018	5.125e-3
m	1192	1150	800-1600	1257.6	5.125e-3

Table 5.5: *Parameters optimised in VDL model (steer ramp).*

Parameter	Original	Init guess	TOC limits	Optimised	Cost
C_{12}	72000	70000	20000-800000	77507.6	2.482e-3
C_{34}	72000	60000	20000-800000	27813.9	2.482e-3
h	0.48	0.5	0.1-2	1.04964	2.482e-3
b	2.24	2.2	0.5-2.47	2.02792	2.482e-3
m	1192	1150	800-1600	941.207	2.482e-3

Table 5.6: *Parameters optimised in VDL model (lane change).*

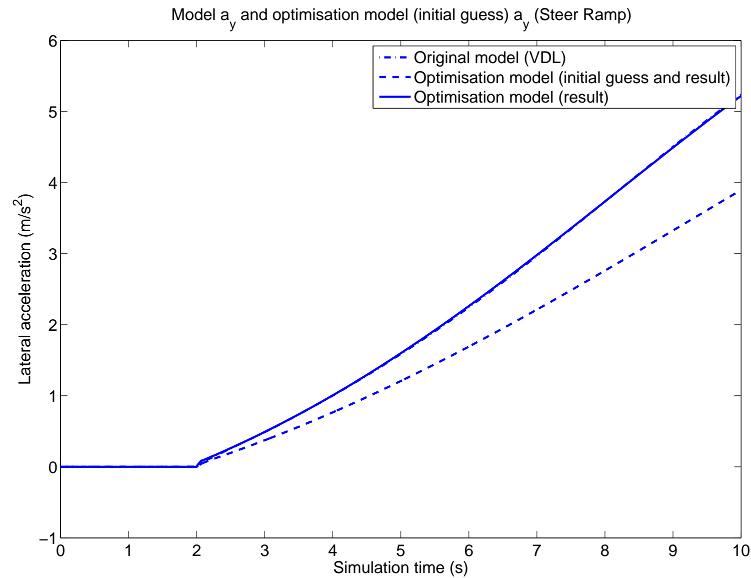


Figure 5.8: *Comparison of lateral acceleration between VDL-, initial guess- and optimised model for a steer ramp.*

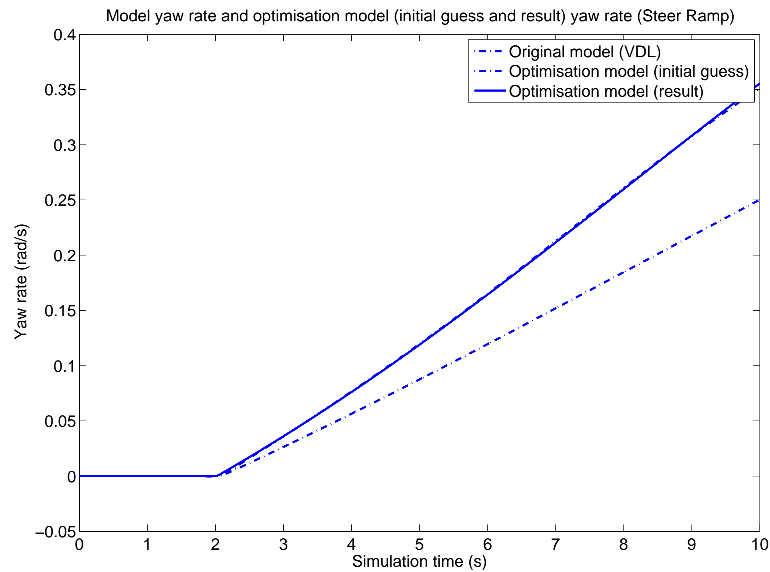


Figure 5.9: *Comparison of yaw rate between VDL-, initial guess- and optimised model for a steer ramp.*

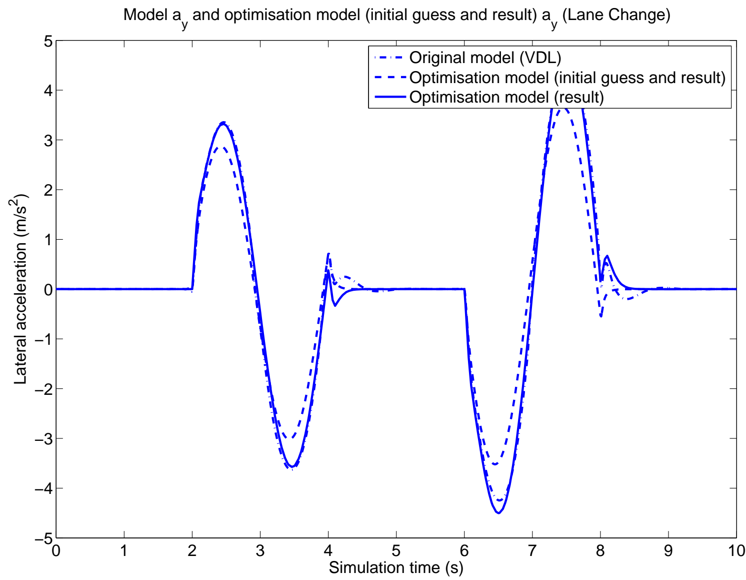


Figure 5.10: *Comparison of lateral acceleration between VDL-, initial guess- and optimised model for a lane change.*

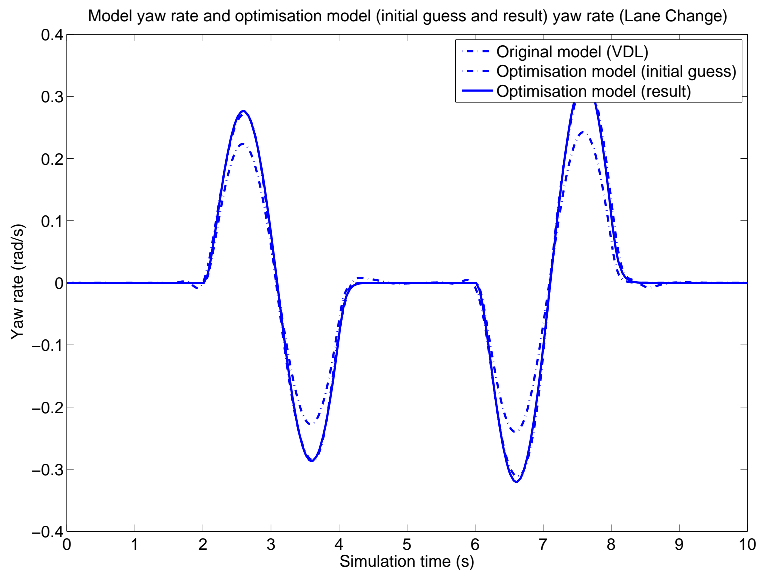


Figure 5.11: *Comparison of yaw rate between VDL-, initial guess- and optimised model for a lane change.*

6

CONCLUSION

6.1 Spline Toolbox

The first part of this thesis was to develop the spline toolbox. This turned out to be a more difficult task than first expected. Different variable types in the programming languages caused problems. As described in the *Curve Fitting with Splines* chapter, three approaches were tried to overcome these problems. The solution that turned out to be the most practical was the wrapper solution. This made it possible to use the FORTRAN source code unchanged, which was a high ranking requirement. Wrapper construction was quite straightforward when the first wrapper was constructed. Difficulties were the matrix transposing, especially the forth dimensional matrix that appeared in the two dimensional spline wrapper.

One drawback with using the PPPack written in FORTRAN77 is that it uses single precision, if more precision is needed a future solution might be to use the FORTRAN90 PPPack that uses double precision.

The resulting package **Splines 1.1** turned out to work as expected in the DYMOLA environment. The practical usage of the package was limited due to the initial lack of compatibility to AMPL. With more practical experience with **Splines 1.1**, issues might arise that can warrant improvements and expansions. The intermediate solutions with MATLAB files generating splines and AMPL splines did it possible to optimise without using the constructed spline package directly.

6.2 Optimisation

6.2.1 General optimisation

The optimisation tools worked very well, for more simple problems the solution was found with a high degree of accuracy. When the problems are more complex the initial guesses are more important for finding a solution. The

initial guess affects the chances of finding a valid solution and the number of iterations required to find the solution, and thus the time to find it.

6.2.2 Vehicle Parameter Optimisation

When optimising parameters in a vehicle model it is important to choose manoeuvres for the models that excite the system adequately to get good parameter values. When optimising a low complexity model to a high complexity model it is important to choose trajectories that are possible for both models to follow. Table 6.1 gives a short overview of typical manoeuvres that are suitable to find certain parameters. These are not all of the manoeuvres possible to perform, but selected common ones. Steady state parameters are for instance mass or tyre constants, dynamic parameters can be inertias or roll dynamics.

Vehicle manoeuvre	States in cost function	Type of parameters in optimisation
Turn with constant speed	a_y, $\ddot{\Psi}$	steady state
Lane change	a_y, $\ddot{\Psi}$, $\dot{\varphi}$	dynamic
Fishhook	a_y, $\ddot{\Psi}$, $\dot{\varphi}$	dynamic

Table 6.1: *Suitable manoeuvres and cost function states for finding different types of parameters*

The time consumed for the optimisation were significant larger with models from VDL. It was also much harder to find reasonable parameters. This was the very last part of this thesis and due to lack of time it has only been addressed briefly.

6.3 Future Work

6.3.1 Splines1.1

As stated earlier, the spline package Splines1.1 is finished, but minor errors or expansion requests might arise when using it more thoroughly.

6.3.2 Vehicle Parameter Optimisation

To optimise parameters for the simple models that were constructed according to Chapter 3 worked very well. It was more difficult to optimise the simple models against the models from VDL, and since the thesis is time limited it was necessary to end the experiments with just a few runs against the VDL. This is an area were much time can be spent in the future. The simple models can be expanded and the choices of manoeuvres can be altered or added to.

The modelling of tires is a topic were much work can be done. The nonlinear tires used in this thesis has been modelled with the Magic Formula, this

can be expanded with longitudinal slip. The Brush model has been modelled but not experimented on.

A

Spline Tools in Modelica

Version 1.1
26-April-2007

This Modelica package contains spline tools to be used in DYMOLA. The subpackage *PPPack* consists of several wrapper functions that call external FORTRAN subroutines, written by Carl De Boor. The structure of the package is presented below in Figure A.1. Functions intended to be used by the user are listed directly under *Splines*. Descriptions of the spline functions are presented in section A.2.

A.1 Examples

The package also includes six examples, listed below. These examples demonstrate the functions of the different splines.

1. Creating cubic-spline (PP-form) and retrieving values with `ppval` Plot with Commands see Figure A.3.

2. This example clearly illustrates the affect of the *smoothing parameter*. The model creates two smooth-splines (PP-form) with different smoothing ability and retrieving values with `ppval`. For comparison with the original data, plot with Command see Figure A.3

3. OptPPSpline consists of three steps:

   - Create optimal knots using `splopt`
   - Calculate B-Spline using `splint`
   - Convert to PP-representation using `b2pp`

   Plot with Command see Figure A.3

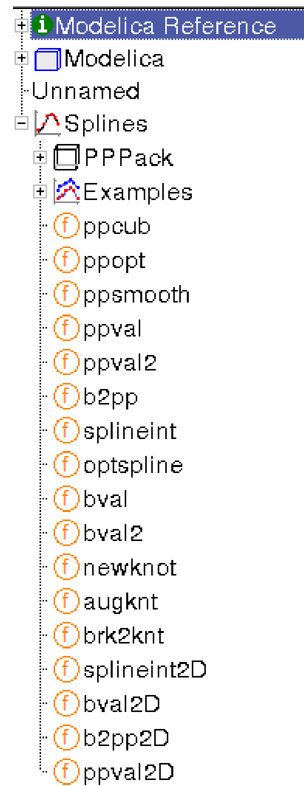


Figure A.1: *The structure of the package. Intended functions to be used are listed under Splines, functions in PPPack are called from the Splines-functions.*

4. Creating spline with interpolation (B-form) and retrieving values with `bval`. Using knots created with `splopt`. Plot with Command see Figure A.3

5. Essentially same as example Splineinterpolation except that everything is done in one step. The spline is created using interpolation (B-form) and values are retrieved with `bval2`. Knots are created with `splopt`.

   Plot with Command see Figure A.3

   Note that not all of the outputs from PPPack are taken care of, this is not necessary for this example..

6. Constructs a two dimensional B-spline, converts it to PP-representation and retrieves values. Uses values from example 17.3 in PGS.

For a control of the complete package, it is recommended to run the script *Test Package* in the model *TestOfPackage* see Figure A.4

Figure A.2: *There are six examples for demonstration purpose included in the package.*

Figure A.3: *To plot results a special command is necessary.*

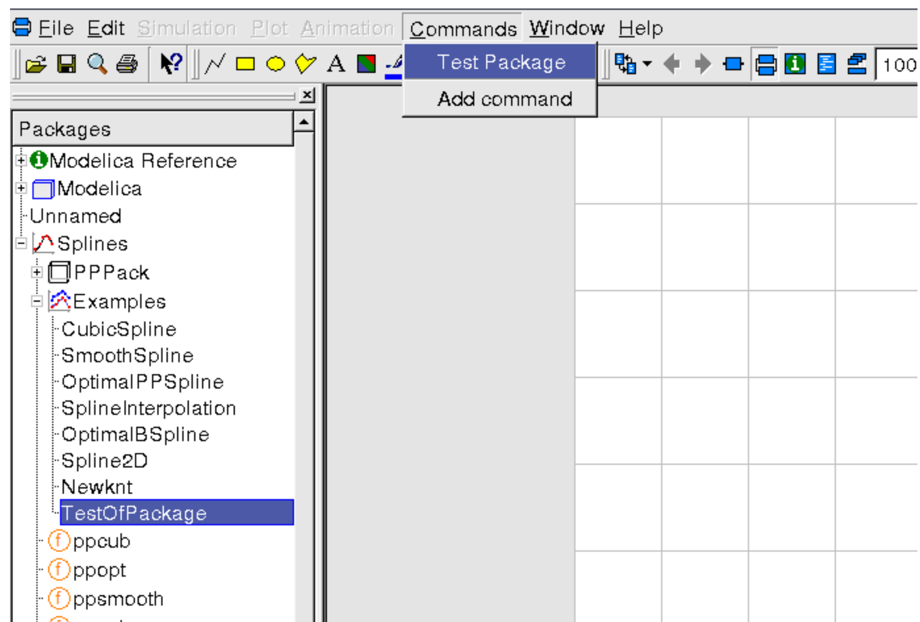


Figure A.4: *A complete test of all examples*

A.2 Spline construction

`ppcub(x,y,icbeg,icend)`

Cubic spline interpolation
Gives a pp-cubic spline with not-a-knot end condition as default.
matlab-spline toolbox corresponding function: `csapi`
Input

- x-values
- y-values
- icbeg
  - ibcbeg = 0 means no boundary condition at tau(1) is given. in this case, the not-a-knot condition is used, i.e. the jump in the third derivative across tau(2) is forced to zero, thus the first and the second cubic polynomial pieces are made to coincide. Default
  - ibcbeg = 1 means that the slope at tau(1) is made to equal c(2,1), supplied by input.
  - ibcbeg = 2 means that the second derivative at tau(1) is made to equal c(2,1), supplied by input.
- ibcend = 0, 1, or 2 has analogous meaning concerning the boundary condition at tau(n), with the additional information taken from c(2,n).

Output

- coef polynomial coefficient

`ppopt(x,y,k)`

Spline interpolant B-spline interpolant, arbitrary order, given data, same as `optspline(x,y,k)` but return a pp-rep
Input

- x-values
- y-values
- order of spline

Output

- pp-function ie. l, coef, break

`ppsmooth(x,y,dy,s)`

Cubic smoothing spline Constructs the cubic smoothing spline f to given data
matlab-spline toolbox corresponding function: `spaps`
Input

- x-values
- y-values
- estimate of uncertainty in data
- upper bound on the discrete weighted mean square distance of the approximation f from the data

Output

- coef polynomial coefficient

`splineint(x,y,t,k)`

B-spline B-spline interpolant, arbitrary order, given data
matlab-spline toolbox corresponding function: `spapi`
Input

- x-values
- y-values
- knot sequence
- order of spline

Output

- the b-coefficients of the interpolant

`optspline(x,y,k)`

B-spline B-spline interpolant, arbitrary order, given data same as `splineint(x,y,t,k)` but calculates the knots t for the optimal recovery scheme of order k
Input

- x-values
- y-values
- order of spline

Output

- the b-coefficients of the interpolant

Evaluation of splines and work on knots and breaks

`ppval(break,coef,x,k,jderiv)`

Pp evaluation
Evaluate value of pp-spline in specified point
`ppval(break,coef,x,k,jderiv)`
Input

- break points
- coefficients
- point to evaluate
- degree of polynomials
- n^{th} derivative in point x

Output

- output value in x, n^{th} derivative

`ppval2(ref,break,coef,x,k,jderiv)`

Pp evaluation
Evaluate value of pp-spline in specified point. This version has better efficiency when several splines are to be evaluated. It has different from `ppval` ability to handle multiple interval index.
Input

- reference number for spline
- break points
- coefficients
- point to evaluate
- degree of polynomials
- n^{th} derivative in point x

Output

- output value in x, n^{th} derivative

`bval(knots,bcoef,x,k,jderiv)`

Spline evaluation b-rep Calculates value at `x` of jderiv-th derivative of spline from b-repr.
Input

- knot sequence
- b-coefficients

- point to evaluate
- degree of polynomials
- n^{th} derivative in point x

Output

- output value in x, n^{th} derivative

`bval2(ref,knots,bcoef,x,k,jderiv)`
Spline evaluation b-rep Calculates value at x of jderiv-th derivative of spline from b-repr.Like `ppval2` it is more efficient than `bval` if multiple splines are to be evaluated alternately.
Input

- reference number for spline
- knot sequence
- b-coefficients
- point to evaluate
- degree of polynomials
- n^{th} derivative in point x

Output

- output value in x, n^{th} derivative

`b2pp(knots,bcoef,k)`
Convert b-spline representation to piecewise polynomial. Converts the b-representation t, bcoef, n, k of some spline into its pp-representation break, coef, l, k .
matlab-spline toolbox corresponding function: `fn2fm` this function has more transform options than b-to-pp.
Input

- knot sequence
- b-coefficients
- degree of polynomials

Output

- l
- pp-coefficient
- breakpoints

`newknot(breaks,coef,lnew)`

New break distribution Generate new knots, optimal locations, given pice-wise polynomial representation
matlab-spline toolbox corresponding function: `newknt`
Input

- breakpoints
- pp-coefficients
- number of intervals into which the interval (a,b) is to besectioned by the new breakpoint sequence `brknew`

Output

- new breakpoint sequence `brknew`
- the coefficient part of the pp-repr

`augknt(breaks,k)`

Augment knot sequence
Places knots in the following way:
$t(1),\ldots,t(n+k) = break(1)$ k times, then $break(i)$, $i = 2\ldots,l$ each once, then $break(l+1)$ k times.
matlab-spline toolbox corresponding function: `augknt`
Input

- breakpoints
- k the order

Output

- t(1),. . . ,t(n+k) the knot sequence
- the dimension of the corresponding spline space of order k

`brk2knt(breaks,k)`

Break to knots
Placement:
$t(1),\ldots,t(n+kpm) = break(1)$ k times, then $break(2),\ldots,break(l)$ each k times, then, finally, $break(l+1)$ k times.
matlab-spline toolbox corresponding function: `brk2knt` Input

- breakpoints
- k the order

Output

- t(1),. . . ,t(n+k) the knot sequence
- $n = l\ldots k$ =dimension of spline(k,t)

`optspline(x,y,k)`

B-spline B-spline interpolant, return a 2D spline of arbitrary order, given data. Input

- x-values
- y-values
- z-values
- knot sequence (x)
- knot sequence (y)
- order of spline (x)
- order of spline (y)

Output

- containing the triangular factorization of the coefficient matrix of the linear system for the b-coefficients of the spline interpolant.
- the b-coefficients of the interpolant
- flag success if = 1 failure if = 2
- reference number

`bval2D(ref,knotx,knoty,bcoef,kx,ky,a, b,jderivA,jderivB)`

2D spline evaluation b-rep Calculates value at a point (a,b) of jderiv-th derivative of spline from 2D b-repr.
Input

- reference number
- knot sequence (x)
- knot sequence (y)
- b-coefficients
- degree of polynomials (x)
- degree of polynomials (y)
- point to evaluate (x)
- point to evaluate (y)
- n^{th} derivative in point a
- n^{th} derivative in point b

Output

- output value in (a,b), n^{th} derivative

`ppval2D(ref,breakx,breaky,coef,kx,ky,a,b,jderivA,jderivB)`

2D spline evaluation pp-rep Calculates value at a point (a,b) of jderiv-th derivative of spline from 2D pp-repr.
Input

- reference number
- break sequence (x)
- break sequence (y)
- coefficients
- degree of polynomials (x)
- degree of polynomials (y)
- point to evaluate (x)
- point to evaluate (y)
- n^{th} derivative in point a
- n^{th} derivative in point b

Output

- output value in (a,b), n^{th} derivative

`b2pp(knotx,knoty,bcoef,kx,ky)`

Convert a 2D b-spline representation to 2D piecewise polynomial. Converts the b-representation tx, ty, bcoef, nx ,ny, kx ,ky of some spline into its pp-representation breakx, breaky, coef, kx, ky .
Input

- knot sequence (x)
- knot sequence (y)
- b-coefficients
- degree of polynomials (x)
- degree of polynomials (y)

Output

- pp-coefficient
- breakpoints (x)
- breakpoints (y)
- reference number

B

Vehicle Models

In this appendix the source code for four of the models are presented.

- One Track model with linear tires
- Two Track model with linear tires
- Two Track model with magic formula tires(pure lateral slip) and load distribution

B.1 One Track with Linear Tyres

```
partial model OneTrack

 parameter Modelica.SIunits.Inertia J_z=2800 "Yaw inertia";
 parameter Modelica.SIunits.Mass m=1550 "Vehicle mass";
 parameter Real C_12=40000 "Front axle side stiffness";
 parameter Real C_34=50000 "Rear axle side stiffness";
 parameter Modelica.SIunits.Length f=1.3
    "Distance from centre of gravity to front axle";
 parameter Modelica.SIunits.Length b=1.4
    "Distance from centre of gravity to rear axle";
 parameter Real i_s(final quantity="Gear ratio")=16 "Steering gain";

 Modelica.SIunits.Velocity v_x(start=1.0) "Longitudinal velocity";
 Modelica.SIunits.Angle delta "Steering wheel angle";
 Modelica.SIunits.Position r_x "Global x position";
 Modelica.SIunits.Position r_y "Global y position";
 Modelica.SIunits.Velocity dr_x "Global x velocity";
 Modelica.SIunits.Velocity dr_y "Global y velocity";
 Modelica.SIunits.Velocity v_y(start=0) "Lateral velocity";
 Modelica.SIunits.Acceleration dv_y;
 /*Derivative of lateral velocity*/
 Modelica.SIunits.Acceleration dv_x;
/*Derivative of longitudinal velocity*/
 Modelica.SIunits.Acceleration a_y "Lateral acceleration";
 Modelica.SIunits.Acceleration a_x "Longitudinal acceleration";
```

```
 Modelica.SIunits.Angle psi "Yaw angle";
 Modelica.SIunits.AngularVelocity dpsi "Yaw velocity";
 Modelica.SIunits.AngularAcceleration ddpsi "Yaw acceleration";
 Modelica.SIunits.Angle delta_12 "Front axle steer angle";
 Modelica.SIunits.Angle beta "Vehicle body slip angle";
 Modelica.SIunits.Angle beta_12 "Front body side slip angle";
 Modelica.SIunits.Angle beta_34 "Rear body side slip angle";
 Modelica.SIunits.Angle alpha_12 "Front wheel side slip angle";
 Modelica.SIunits.Angle alpha_34 "Rear wheel side slip angle";
 Modelica.SIunits.Force f_12 "Front wheels lateral force";
 Modelica.SIunits.Force f_21 "Front wheels longitudinal force";
 Modelica.SIunits.Force f_34 "Rear wheel lateral force";
 Modelica.SIunits.Force f_43 "Rear wheel longitudinal force";

equation
 assert(v_x>0.01, "Longitudinal velocity (v_x) is to low");

 /* Slip */
 i_s*delta_12=delta;
 beta_12 = atan((v_y+dpsi*f)/v_x);
 alpha_12 = beta_12-delta_12;

 beta_34 = atan((v_y-dpsi*b)/v_x);
 alpha_34 = beta_34;

 beta = atan(v_y/v_x);

 /* Tyre forces */
 f_12=-C_12*alpha_12;
 f_34=-C_34*alpha_34;

 /* Kinematics */
 der(psi)=dpsi;
 der(dpsi)=ddpsi;
 der(r_x)=dr_x;
 der(r_y)=dr_y;
 der(v_y)=dv_y;
 der(v_x)=dv_x;

 dr_x = v_x*cos(psi) - v_y*sin(psi) "Coordinate transform";
 dr_y = v_x*sin(psi) + v_y*cos(psi) "Coordinate transform";

 a_y = dv_y+dpsi*v_x;
 a_x = dv_x+dpsi*v_y;

 /* Force balance*/
 m*a_x= f_21*cos(delta_12) - f_12*sin(delta_12) + f_43;
 m*a_y= f_12*cos(delta_12) + f_21*sin(delta_12) + f_34;
 J_z*ddpsi = f*(f_12*cos(delta_12)+f_21*sin(delta_12)) - b*f_34;

 f_43=0;

end OneTrack;
```

B.2 Two Track Linear Tyres

```
partial model TwoTrack

 import SI = Modelica.SIunits;
 parameter SI.Inertia J_z=2800 "Yaw inertia";
 parameter SI.Mass m=1550 "Vehicle mass";
```

```
 parameter Real C_12=40000 "Front axle side stiffness";
 parameter Real C_34=50000 "Rear axle side stiffness";
 parameter SI.Length f=1.3;
 /*Distance from centre of gravity to front axle*/
 parameter SI.Length b=1.4;
 /*Distance from centre of gravity to rear axle*/
 parameter SI.Length t_w=1.8 "Distance between tyre centers";
 parameter SI.Length h=0.55 "height of CG in z";
 parameter Real k_f = 12 "front axel vridstyvhet";
 parameter Real k_r = 10 "rear axel vridstyvhet";
 Real tau_x_f "front axel moment";
 Real tau_x_r "rear axel moment";

 parameter Real i_s(final quantity="Gear ratio")=16;
 /*Steering gain*/

 SI.Velocity v_x(start=0.02) "Longitudinal velocity";
 SI.Angle delta "Steering wheel angle";
 SI.Position r_x "Global x position";
 SI.Position r_y "Global y position";
 SI.Velocity dr_x "Global x velocity";
 SI.Velocity dr_y "Global y velocity";
 SI.Velocity v_y(start=0) "Lateral velocity";
 SI.Acceleration dv_y "Derivative of lateral velocity";
 SI.Acceleration dv_x "Derivative of longitudinal velocity";
 SI.Acceleration a_y "Lateral acceleration";
 SI.Acceleration a_x "Longitudinal acceleration";
 SI.Angle psi "Yaw angle";
 SI.AngularVelocity dpsi "Yaw velocity";
 SI.AngularAcceleration ddpsi "Yaw acceleration";
 SI.Angle delta_12 "Front axle steer angle";
 SI.Angle beta "Vehicle body slip angle";
 SI.Angle beta_12 "Front body side slip angle";
 SI.Angle beta_34 "Rear body side slip angle";
 SI.Angle alpha_12 "Front wheel side slip angle";
 SI.Angle alpha_34 "Rear wheel side slip angle";

 SI.Force f_1x;
 SI.Force f_1y;
 SI.Force f_1z;

 SI.Force f_2x;
 SI.Force f_2y;
 SI.Force f_2z;

 SI.Force f_3x;
 SI.Force f_3y;
 SI.Force f_3z;

 SI.Force f_4x;
 SI.Force f_4y;
 SI.Force f_4z;

 constant SI.Acceleration g = 9.82;

equation
 assert(v_x>0.01, "Longitudinal velocity (v_x) is to low");

 /* Slip */
 i_s*delta_12=delta;
 beta_12 = atan((v_y+dpsi*f)/v_x);
 alpha_12 = beta_12-delta_12;
```

```
    beta_34 = atan((v_y-dpsi*b)/v_x);
    alpha_34 = beta_34;

    beta = atan(v_y/v_x);

    /* Tyre forces */
    //f_12=-C_12*alpha_12;
    f_1y=-C_12*alpha_12;
    f_2y=-C_12*alpha_12;

    //f_34=-C_34*alpha_34;
    f_3y=-C_34*alpha_34;
    f_4y=-C_34*alpha_34;

    /* Kinematics */
    der(psi)=dpsi;
    der(dpsi)=ddpsi;
    der(r_x)=dr_x;
    der(r_y)=dr_y;
    der(v_y)=dv_y;
    der(v_x)=dv_x;

    dr_x = v_x*cos(psi) - v_y*sin(psi) "Coordinate transform";
    dr_y = v_x*sin(psi) + v_y*cos(psi) "Coordinate transform";

    a_y = dv_y+dpsi*v_x;
    a_x = dv_x+dpsi*v_y;

    /*Force balance*/
    m*g = f_1z + f_2z + f_3z + f_4z;
    /*z force equality*/
    m*a_y*h = f_1z*t_w/2 -f_2z*t_w/2 +f_3z*t_w/2 -f_4z*t_w/2;
    /*moment equality around x axel*/
    m*a_x*h = -f_1z*f -f_2z*f  +f_3z*b +f_4z*b;
    /*moment equality around y axel*/

    tau_x_f*k_r = tau_x_r*k_f;          //relation when phi_f = phi_r
    tau_x_f = f_1z*t_w/2 -f_2z*t_w/2;
    tau_x_r = f_3z*t_w/2 -f_4z*t_w/2;

    m*a_x= f_1x*cos(delta_12) +f_2x*cos(delta_12) -f_1y*sin(delta_12)
    -f_2y*sin(delta_12) +f_3x +f_4x;

    m*a_y= f_1y*cos(delta_12) +f_2y*cos(delta_12) +f_1x*sin(delta_12)
    +f_2x*sin(delta_12) +f_3y +f_4y;

    J_z*ddpsi = f*(f_1y*cos(delta_12) +f_2y*cos(delta_12)
    +f_1x*sin(delta_12) +f_2x*sin(delta_12)) -b*(f_3y +f_4y);

    f_3x = 0;
    f_4x = 0;
    f_1x = f_2x;

end TwoTrack;
```

B.3 Two Track Magic Formula and Load Distribution

```
partial model TwoTrackRollMagicFormula "Individual tyre force input"
```

```
 import SI = Modelica.SIunits;

parameter SI.Inertia J_z=1800 "Yaw inertia";
parameter SI.Inertia J_phi=500 "Roll inertia";
parameter SI.Mass m=1100 "Vehicle mass";
parameter SI.Length f=L-b "Distance from centre of gravity to front axle";
parameter SI.Length b=1.17 "Distance from centre of gravity to rear axle";
parameter SI.Length L=2.47 "Vehicle length";
parameter SI.Length t_w=1.6 "Distance between tyre centers";
parameter SI.Length h=0.25 "height of CG in z";
parameter Real h_1 = 0.30 "height of front roll center";
parameter Real h_2 = 0.35 "height of rear roll center";
parameter Real c_phi1 = 40000 "front roll stiffness";
parameter Real c_phi2 = 25000 "rear roll stiffness";
parameter Real i_s(final quantity="Gear ratio")=15 "Steering gain";
parameter SI.Damping d_phi = 7000 "Roll damping";

parameter Real k=1 "Gain";
parameter SI.Time T=0.01 "Time Constant";
parameter Real mu_f=1 "friction front";
parameter Real mu_r=1 "friction rear";

/*Magic formula coefficents*/
Real B_1 "Stiffness factor";
parameter Real C_1=1.2 "Shape factor";
Real D_1 "Peak factor";
Real E_1 "Curvature factor";

Real B_3 "Stiffness factor";
parameter Real C_3=1.2 "Shape factor";
Real D_3 "Peak factor";
Real E_3 "Curvature factor";

/*Magic formula coefficents*/
Real B_2 "Stiffness factor";
parameter Real C_2=1.2 "Shape factor";
Real D_2 "Peak factor";
Real E_2 "Curvature factor";

Real B_4 "Stiffness factor";
parameter Real C_4=1.2 "Shape factor";
Real D_4 "Peak factor";
Real E_4 "Curvature factor";

/*Parameters in MF*/
parameter Real a_31=35000;
parameter Real a_32=35000;
parameter Real a_33=40000;
parameter Real a_34=40000;

parameter Real a_41=5000;
parameter Real a_42=5000;
parameter Real a_43=5000;
parameter Real a_44=5000;

parameter Real a_61=-0.0006;
parameter Real a_62=-0.0006;
parameter Real a_63=-0.0006;
parameter Real a_64=-0.0006;
SI.Acceleration g = Modelica.Constants.g_n "Acceleration du to gravity";

SI.Distance h_roll "Distance from CG to roll center";
```

```
  SI.Angle phi "Roll angle";
  SI.Velocity v_x(start=10) "Longitudinal velocity";
  SI.Angle delta "Steering wheel angle";

  SI.Position r_x "Global x position";
  SI.Position r_y "Global y position";
  SI.Velocity dr_x "Global x velocity";
  SI.Velocity dr_y "Global y velocity";

  SI.Velocity v_y "Lateral velocity";
  SI.Acceleration dv_y "Derivative of lateral velocity";
  SI.Acceleration dv_x "Derivative of longitudinal velocity";
  SI.Acceleration a_y( start=0.01) "Lateral acceleration";
  SI.Acceleration a_x "Longitudinal acceleration";
  SI.Angle psi "Yaw angle";
  SI.AngularVelocity dpsi "Yaw velocity";
  SI.AngularAcceleration ddpsi "Yaw acceleration";
  SI.AngularVelocity dphi;
  SI.AngularAcceleration ddphi;

  SI.Angle delta_12 "Front axle steer angle";
  SI.Angle beta "Vehicle body slip angle";
  SI.Angle beta_12 "Front body side slip angle";
  SI.Angle beta_34 "Rear body side slip angle";
  SI.Angle alpha_12 "Front wheel side slip angle";
  SI.Angle alpha_34 "Rear wheel side slip angle";

  SI.Force f_1x "longitudinalForce";
  SI.Force f_1y;
  SI.Force f_1u;
  SI.Force f_1z;

  SI.Force f_2x "longitudinalForce";
  SI.Force f_2y;
  SI.Force f_2u;
  SI.Force f_2z;

  SI.Force f_3x "longitudinalForce";
  SI.Force f_3y;
  SI.Force f_3u;
  SI.Force f_3z;

  SI.Force f_4x "longitudinalForce";
  SI.Force f_4y;
  SI.Force f_4u;
  SI.Force f_4z;

  SI.Force delta_F_z1 "load transfer front";
  SI.Force delta_F_z2 "load transfer rear";

  Real BCD1 "slip stiffnes at zero slip";
  Real BCD2 "slip stiffnes at zero slip";
  Real BCD3 "slip stiffnes at zero slip";
  Real BCD4 "slip stiffnes at zero slip";

equation
  assert(v_x>0.01, "Longitudinal velocity (v_x) is to low");

  /* Slip */
  i_s*delta_12=delta;
  beta_12 = atan((v_y+dpsi*f)/v_x);
  alpha_12 = beta_12-delta_12;
```

```
beta_34 = atan((v_y-dpsi*b)/v_x);
alpha_34 = beta_34;

beta = atan(v_y/v_x);

/* Tyre forces */
f_1u=-D_1*sin(C_1*atan(B_1*alpha_12-E_1*(B_1*alpha_12-atan(B_1*alpha_12))));
f_2u=-D_2*sin(C_2*atan(B_2*alpha_12-E_2*(B_2*alpha_12-atan(B_2*alpha_12))));
f_3u=-D_3*sin(C_3*atan(B_3*alpha_34-E_3*(B_3*alpha_34-atan(B_3*beta_34))));
f_4u=-D_4*sin(C_4*atan(B_4*alpha_34-E_4*(B_4*alpha_34-atan(B_4*beta_34))));

/*First order filter models elasticity in tires*/
der(f_1y) = (k*f_1u - f_1y)/T;
der(f_2y) = (k*f_2u - f_2y)/T;
der(f_3y) = (k*f_3u - f_3y)/T;
der(f_4y) = (k*f_4u - f_4y)/T;

/*Peak factor*/
D_1=mu_f*f_1z;
D_2=mu_f*f_2z;
D_3=mu_r*f_3z;
D_4=mu_r*f_4z;


/*slip stiffness*/
BCD1=a_31*sin(2*atan(f_1z/a_41));
BCD2=a_32*sin(2*atan(f_2z/a_42));
BCD3=a_33*sin(2*atan(f_3z/a_43));
BCD4=a_34*sin(2*atan(f_4z/a_44));

/*Curvature factor*/
E_1=a_61*f_1z;
E_2=a_62*f_2z;
E_3=a_63*f_3z;
E_4=a_64*f_4z;

/*Stiffness factor*/
B_1=BCD1/(C_1*D_1);
B_2=BCD2/(C_2*D_2);
B_3=BCD3/(C_3*D_3);
B_4=BCD4/(C_4*D_4);

/* Kinematics */
der(psi)=dpsi;
der(dpsi)=ddpsi;
der(r_x)=dr_x;
der(r_y)=dr_y;
der(v_y)=dv_y;
der(v_x)=dv_x;
dphi = der(phi);
ddphi = der(dphi);

dr_x = v_x*cos(psi) - v_y*sin(psi) "Coordinate transform";
dr_y = v_x*sin(psi) + v_y*cos(psi) "Coordinate transform";

a_y = dv_y+dpsi*v_x;
a_x = dv_x+dpsi*v_y;

/*Load transfer*/
delta_F_z1 =(1/t_w)*(c_phi1/(c_phi1+c_phi2-m*g*cos(phi)*h_roll)*h_roll
+ b/(f+b)*h_1)*m*a_y;
```

```
 delta_F_z2 =(1/t_w)*(c_phi2/(c_phi1+c_phi2-m*g*cos(phi)*h_roll)*h_roll
+ f/(f+b)*h_2)*m*a_y;

        /*Normal tyre force            force x-direction     inertia*/
 f_1z = b/(2*(f+b))*m*g - delta_F_z1 -h/(2*(f+b))*m*a_x
 + J_phi*ddphi/(t_w/2);
 f_2z = b/(2*(f+b))*m*g + delta_F_z1 -h/(2*(f+b))*m*a_x
 - J_phi*ddphi/(t_w/2);
 f_3z = f/(2*(f+b))*m*g - delta_F_z2 +h/(2*(f+b))*m*a_x
 + J_phi*ddphi/(t_w/2);
 f_4z = f/(2*(f+b))*m*g + delta_F_z2 +h/(2*(f+b))*m*a_x
 - J_phi*ddphi/(t_w/2);

 /*Height from roll centre to CG */
 h_roll = (h - ((h_2-h_1)/(f+b)*f + h_1));


 /*Roll angle dynamics*/
 J_phi*ddphi+d_phi*dphi+(c_phi1+c_phi2)*phi+h_roll*m*a_y=0;

 /*Force balance*/
 m*a_x= f_1x*cos(delta_12) +f_2x*cos(delta_12) -f_1y*sin(delta_12)
-f_2y*sin(delta_12) +f_3x +f_4x;

 m*a_y= f_1y*cos(delta_12) +f_2y*cos(delta_12) +f_1x*sin(delta_12)
+f_2x*sin(delta_12) +f_3y +f_4y;

 J_z*ddpsi = f*(f_1y*cos(delta_12) +f_2y*cos(delta_12)
+f_1x*sin(delta_12) +f_2x*sin(delta_12))
             - b*(f_3y +f_4y);


end TwoTrackRollMagicFormula;
```

C

AMPL Spline Code

C.1 funcadd.c

```
/****************************************************************
Copyright (C) 1997-1998 Lucent Technologies
All Rights Reserved

Permission to use, copy, modify, and distribute this software and
its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that the copyright notice and this
permission notice and warranty disclaimer appear in supporting
documentation, and that the name of Lucent or any of its entities
not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF
THIS SOFTWARE.
****************************************************************/

#include <stdlib.h>
#include "math.h"       /* for sqrt */
#include "funcadd.h"    /* includes "stdio1.h" */
#include "splines.h"

char *ix_details_ASL[] = {0};    /* no -i command-line option */

#define N 5
#include "TwoTrackFiRoll.h"

static real Bsp_f1z(arglist *al) {
```

```
  double x = al->ra[0];
  int i,j;
  int n = f_1z_n;
  int k = 7; //order of the spline

  AmplExports *ae = al->AE; // for fprintf and strtod
  static int init_done = 0;

  static double *t; // knots'en
  static int iflag;
  static double *bcoef;
  static int ref;
  float tW[n+k];
  float bcoefW[n];

  if (init_done == 0){
    init_done = 1;
    t = (double*)calloc(n+k,sizeof(double));
    float scratch[(n-k)*(2*k+3)+5*k+3];
    float tauW[n];
    float tW[n+k];
    sploptwc(f_1z_x,n,k,scratch,t,iflag,tauW,tW);

    bcoef = (double*)calloc(n,sizeof(double));
    double q[(2*k-1)*n];
    float gtauW[n];
    float qW[(2*k-1)*n];
    ref = splintwc(f_1z_x,f_1z_y,t,n,k,q,bcoef,iflag,tauW,
                   gtauW,tW,qW,bcoefW);
  } //end if init

  float xW;
  float yW;
  double res;
  res = bvalue2wc(t,bcoef,n,k,x,0,tW,bcoefW,xW,yW,ref);

  if (al->derivs) {
    *al->derivs = bvalue2wc(t,bcoef,n,k,x,1,tW,
                            bcoefW,xW,yW,ref);
    if (al->hes) {
      *al->hes = bvalue2wc(t,bcoef,n,k,x,2,tW,
                           bcoefW,xW,yW,ref);
    }
  }
  return res;
}

static real Bsp_f2z(arglist *al) {
  double x = al->ra[0];
  int i,j;
  int n = f_2z_n;
  int k = 7; //order of the spline

  AmplExports *ae = al->AE; // for fprintf and strtod
  static int init_done = 0;

  static double *t; // knots'en
  static int iflag;
  static double *bcoef;
  static int ref;
  float tW[n+k];
  float bcoefW[n];
```

```
  if (init_done == 0){
    init_done = 1;
    t = (double*)calloc(n+k,sizeof(double));
    float scratch[(n-k)*(2*k+3)+5*k+3];
    float tauW[n];
    float tW[n+k];
    sploptwc(f_2z_x,n,k,scratch,t,iflag,tauW,tW);

    bcoef = (double*)calloc(n,sizeof(double));
    double q[(2*k-1)*n];
    float gtauW[n];
    float qW[(2*k-1)*n];
    ref = splintwc(f_2z_x,f_2z_y,t,n,k,q,bcoef,iflag,tauW,
                   gtauW,tW,qW,bcoefW);
  } //end if init

  float xW;
  float yW;
  double res;
  res = bvalue2wc(t,bcoef,n,k,x,0,tW,bcoefW,xW,yW,ref);

  if (al->derivs) {
    *al->derivs = bvalue2wc(t,bcoef,n,k,x,1,tW,
                            bcoefW,xW,yW,ref);
    if (al->hes) {
      *al->hes = bvalue2wc(t,bcoef,n,k,x,2,tW,
                           bcoefW,xW,yW,ref);
    }
  }
  return res;
}

static real Bsp_f3z(arglist *al) {
  double x = al->ra[0];
  int i,j;
  int n = f_3z_n;
  int k = 7; //order of the spline

  AmplExports *ae = al->AE; // for fprintf and strtod
  static int init_done = 0;

  static double *t; // knots'en
  static int iflag;
  static double *bcoef;
  static int ref;
  float tW[n+k];
  float bcoefW[n];

  if (init_done == 0){
    init_done = 1;
    t = (double*)calloc(n+k,sizeof(double));
    float scratch[(n-k)*(2*k+3)+5*k+3];
    float tauW[n];
    float tW[n+k];
    sploptwc(f_3z_x,n,k,scratch,t,iflag,tauW,tW);

    bcoef = (double*)calloc(n,sizeof(double));
    double q[(2*k-1)*n];
    float gtauW[n];
    float qW[(2*k-1)*n];
    ref = splintwc(f_3z_x,f_3z_y,t,n,k,q,bcoef,iflag,tauW,
```

```
                    gtauW,tW,qW,bcoefW);
  } //end if init

  float xW;
  float yW;
  double res;
  res = bvalue2wc(t,bcoef,n,k,x,0,tW,bcoefW,xW,yW,ref);

  if (al->derivs) {
    *al->derivs = bvalue2wc(t,bcoef,n,k,x,1,tW,
                            bcoefW,xW,yW,ref);
    if (al->hes) {
      *al->hes = bvalue2wc(t,bcoef,n,k,x,2,tW,
                           bcoefW,xW,yW,ref);
    }
  }
  return res;
}
static real Bsp_f4z(arglist *al) {
  double x = al->ra[0];
  int i,j;
  int n = f_4z_n;
  int k = 7; //order of the spline

  AmplExports *ae = al->AE; // for fprintf and strtod
  static int init_done = 0;

  static double *t; // knots'en
  static int iflag;
  static double *bcoef;
  static int ref;
  float tW[n+k];
  float bcoefW[n];

  if (init_done == 0){
    init_done = 1;
    t = (double*)calloc(n+k,sizeof(double));
    float scratch[(n-k)*(2*k+3)+5*k+3];
    float tauW[n];
    float tW[n+k];
    sploptwc(f_4z_x,n,k,scratch,t,iflag,tauW,tW);

    bcoef = (double*)calloc(n,sizeof(double));
    double q[(2*k-1)*n];
    float gtauW[n];
    float qW[(2*k-1)*n];
    ref = splintwc(f_4z_x,f_4z_y,t,n,k,q,bcoef,iflag,tauW,
                   gtauW,tW,qW,bcoefW);
  } //end if init

  float xW;
  float yW;
  double res;
  res = bvalue2wc(t,bcoef,n,k,x,0,tW,bcoefW,xW,yW,ref);

  if (al->derivs) {
    *al->derivs = bvalue2wc(t,bcoef,n,k,x,1,tW,
                            bcoefW,xW,yW,ref);
    if (al->hes) {
      *al->hes = bvalue2wc(t,bcoef,n,k,x,2,tW,
                           bcoefW,xW,yW,ref);
    }
```

```
  }
  return res;
}

static real Bspline_a_x(arglist *al) {
  double x = al->ra[0];
  int i,j;
  int n = Sp_a_x_n;
  int k = 7; //order of the spline

  AmplExports *ae = al->AE; // for fprintf and strtod
  static int init_done = 0;

  static double *t; // knots'en
  static int iflag;
  static double *bcoef;
  static int ref;
  float tW[n+k];
  float bcoefW[n];

  if (init_done == 0){
    init_done = 1;
    t = (double*)calloc(n+k,sizeof(double));
    float scratch[(n-k)*(2*k+3)+5*k+3];
    float tauW[n];
    float tW[n+k];
    sploptwc(Sp_a_x_x,n,k,scratch,t,iflag,tauW,tW);
    bcoef = (double*)calloc(n,sizeof(double));
    double q[(2*k-1)*n];
    float gtauW[n];
    float qW[(2*k-1)*n];
    ref = splintwc(Sp_a_x_x,Sp_a_x_y,t,n,k,q,bcoef,iflag,tauW,
                   gtauW,tW,qW,bcoefW);
  } //end if init

  float xW;
  float yW;
  double res;
  res = bvalue2wc(t,bcoef,n,k,x,0,tW,bcoefW,xW,yW,ref);

  if (al->derivs) {
    *al->derivs = bvalue2wc(t,bcoef,n,k,x,1,tW,
                            bcoefW,xW,yW,ref);
    if (al->hes) {
      *al->hes = bvalue2wc(t,bcoef,n,k,x,2,tW,
                           bcoefW,xW,yW,ref);
    }
  }
  return res;
}

static real Bspline_a_y(arglist *al) {
  double x = al->ra[0];
  int i,j;
  int n = Sp_a_y_n;
  int k = 7; //order of the spline

  AmplExports *ae = al->AE; // for fprintf and strtod
  static int init_done = 0;

  static double *t; // knots'en
  static int iflag;
```

```
  static double *bcoef;
  static int ref;
  float tW[n+k];
  float bcoefW[n];

  //  fprintf(stderr, "knutar!\n");
  if (init_done == 0){
    init_done = 1;
    t = (double*)calloc(n+k,sizeof(double));
    float scratch[(n-k)*(2*k+3)+5*k+3];
    float tauW[n];
    float tW[n+k];
    sploptwc(Sp_a_y_x,n,k,scratch,t,iflag,tauW,tW);

    bcoef = (double*)calloc(n,sizeof(double));
    double q[(2*k-1)*n];
    float gtauW[n];
    float qW[(2*k-1)*n];
    ref = splintwc(Sp_a_y_x,Sp_a_y_y,t,n,k,q,bcoef,iflag,tauW,
                   gtauW,tW,qW,bcoefW);
  } //end if init

  float xW;
  float yW;
  double res;
  res = bvalue2wc(t,bcoef,n,k,x,0,tW,bcoefW,xW,yW,ref);

  if (al->derivs) {
    *al->derivs = bvalue2wc(t,bcoef,n,k,x,1,tW,
                            bcoefW,xW,yW,ref);
    if (al->hes) {
      *al->hes = bvalue2wc(t,bcoef,n,k,x,2,tW,
                           bcoefW,xW,yW,ref);
    }
  }
  return res;
}

static real Spf_a_x(arglist *al) {
  double x = al->ra[0];
  int i,j;
  int n = Sp_a_x_n;

  AmplExports *ae = al->AE; // for fprintf and strtod

  static int init_done = 0;

  static double *c;
  static float *bcW;
  static float *tauW;


  if(init_done == 0){
    init_done = 1;

    // Coefficient data
    double bc[4*n];

    c = (double*)calloc((n-1)*4,sizeof(double));
    bcW = (float*)calloc(4*n,sizeof(float));
    tauW = (float*)calloc(n,sizeof(float));
    if (c == NULL || bcW == NULL || tauW == NULL) {
```

```
      fprintf(stderr, "Couldn't allocate memory\n");
      exit(EXIT_FAILURE);
    }
    // Initialize
    for (i=0;i<n;i++) {
      bc[i] = Sp_a_x_y[i];
    }

    int ibcbeg = 0;
    int ibcend = 0;

    cubsplwc(Sp_a_x_x, bc, n, ibcbeg, ibcend, tauW, bcW);

    for (i=0;i<4;i++) {
      for (j=0;j<n-1;j++) {
        c[i*(n-1)+j] = bc[i*n+j];
      }
    }
  }

  float xW;
  double res;
  res = ppvaluwc(Sp_a_x_x,c,n-1,4,x,0, bcW, tauW, xW);

  if (al->derivs) {
    *al->derivs = ppvaluwc(Sp_a_x_x,c,n-1,4,
                           x,1, bcW, tauW, xW);
    if (al->hes) {
      *al->hes = ppvaluwc(Sp_a_x_x,c,n-1,4,
                          x,2, bcW, tauW, xW);
    }
  }
  return res;
}

void funcadd(AmplExports *ae){
        /* Insert calls on addfunc here... */
        /* Arg 3, called argtype, can be 0 or 1:
         *      0 ==> force all arguments to be numeric
         *      1 ==> pass both symbolic and numeric arguments.
         *
         * Arg 4, called nargs, is interpretted as follows:
         *      >=  0 ==> the function has exactly nargs arguments
         *      <= -1 ==> the function has >= -(nargs+1) arguments.
         *
         * Arg 5, funcinfo, is passed to the functions in
         * struct arglist;
         *      it is not used in these examples, so we just pass 0.
         */
        addfunc("Spf_a_x", (ufunc*)Spf_a_x, 0, 1 , 0);
        addfunc("Bspline_a_y", (ufunc*)Bspline_a_y, 0, 1 , 0);
        addfunc("Bspline_a_x", (ufunc*)Bspline_a_x, 0, 1 , 0);
        addfunc("Bsp_f1z", (ufunc*)Bsp_f1z, 0, 1 , 0);
        addfunc("Bsp_f2z", (ufunc*)Bsp_f2z, 0, 1 , 0);
        addfunc("Bsp_f3z", (ufunc*)Bsp_f3z, 0, 1 , 0);
        addfunc("Bsp_f4z", (ufunc*)Bsp_f4z, 0, 1 , 0);
        }
```

C.2 Roll.run

```
reset;

reload amplfunc.dll;
function Bspline_a_x;
function Bspline_a_y;
function Bsp_f1z;
function Bsp_f2z;
function Bsp_f3z;
function Bsp_f4z;

model TwoTrackFiRoll.CalibrateCarFi_opt.mod;
model TwoTrackFiRoll.CalibrateCarFi_opt.InitialGuess.mod;
model TwoTrackFiRoll.CalibrateCarFi_opt.SquareProblemCost.mod;
model TwoTrackFiRoll.CalibrateCarFi_opt.Constraint.mod;

data TwoTrackFiRoll.CalibrateCarFi_opt.dat;
data TwoTrackFiRoll.CalibrateCarFi_opt.InitialGuess.dat;
option  solver "/work/jakesson/software_tools/Ipopt/Ipopt-3.2.0/
                CoinIpopt/bin/ipopt";
#option solver "/home/jakesson/work/software_tools/Ipopt/
                Ipopt-3.2.0/CoinIpopt/bin/ipopt";
option  ipopt_options "max_iter = 10000 tol=1e-5";
solve;
include TwoTrackFiRoll.CalibrateCarFi_opt.GenLogFile.run;

model Roll.Cost.mod;
solve;
include TwoTrackFiRoll.CalibrateCarFi_opt.GenLogFile.run;
display carFi_J_z;
```

C.3 Roll.Cost.mod

```
redeclare minimize COST: (sum{_i in FE} (TIME*_H[_i]*sum{_j in CP}
 (((carFi_f_1z[_i,_j]-Bsp_f1z(time[_i,_j])))^2+(carFi_f_2z[_i,_j]
-Bsp_f2z(time[_i,_j]))^2+(carFi_f_3z[_i,_j]-
Bsp_f3z(time[_i,_j]))^2+(carFi_f_4z[_i,_j]-
Bsp_f4z(time[_i,_j]))^2)*_w[_j]));
```

BIBLIOGRAPHY

[AMPL®, 2007] AMPL®(2007). A modeling language for mathematical programming. http://www.ampl.com.

[Biegler et al., 2001] Biegler, L. T., Cervantes, A. M., and Wächter, A. (2001). Advances in simultaneous strategies for dynamic process optimization.

[de Boor, 1978] de Boor, C. (1978). *A Practical Guide to Splines*. Springer-Verlag, New York, Heidelberg, Berlin.

[de Boor, 2007] de Boor, C. (2007). *Spline Toolbox 3 User's Guide*. www.mathworks.com.

[Heath, 2002] Heath, M. T. (2002). *Scientific Computing, An Introductory Survey, Second Edition*. McGraw-Hill, New York.

[IPOPT, 2007] IPOPT (2007). Interior point optimization. https://projects.coin-or.org/Ipopt.

[Modelon, 2007] Modelon (2007). http://www.modelon.se.

[Pacejka, 2002] Pacejka, H. B. (2002). *Tire and Vehicle Dynamics*. Butterworth-Heinemann Ltd, Oxford.

[Wennerström et al., 2005] Wennerström, E., Nordmark, S., and Thorvald, B. (2005). *Fordonsdynamik*. Kungliga Tekniska Högskolan Avd. Fordonsdynamik, KTH Farkost och flyg, Stockholm.

[Åkesson Johan, 2007] Åkesson Johan (2007). *The Optimica Compiler 0.3 Users's Guide*.