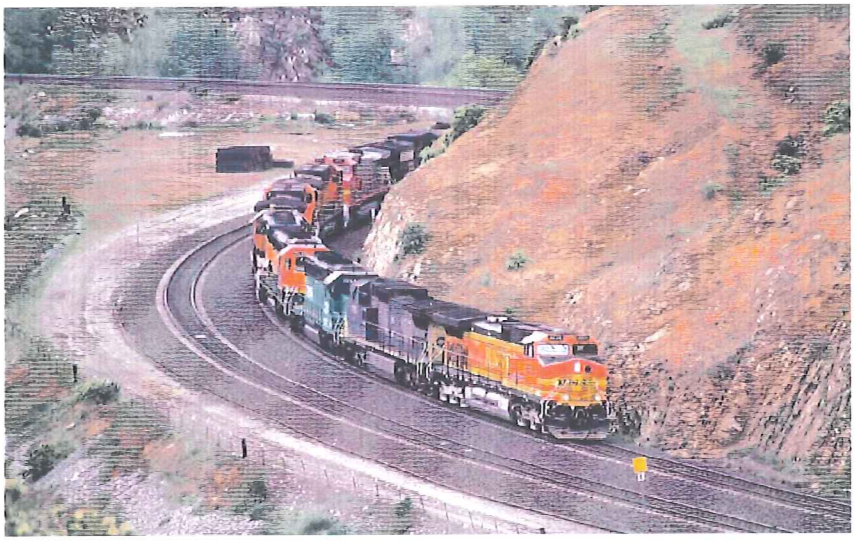
Curving Dynamics of Railway Vehicles

Dan Erik Petersen
c973539

Mark Hoffmann
c973500



Burlington Northern Santa Fe Railway.

13-Weeks Course, 15th May 2002

Supervisor: True, Hans
IMM: Department of Mathematical Modelling
DTU: Technical University of Denmark

Abstract

This project establishes a mathematical model for the dynamics of a railcar bogie and its relevant section of the railcar. The model is based on Cooperrider's bogie. The model accomodates the possibility of running the train through a curve. Furthermore, the rail track can be superelevated.

The critical rail-wheel contact physics are modelled through Hertz' theory proposed in 1882, where wheel-rail deformation are modelled as penetration of the wheel into the rail. Tangential forces are calculated through the method presented by Shen, Hedrick and Elkins.

The implemented program is briefly documented and explained, before ultimately presenting results achieved. Various investigations on the effect of superelevation, curve radius and bogie velocity as parameters has been executed. Special emphasis has been placed on the lowering of the the critical (nonlinear) velocity of the bogie when negotiating a curve.

The results achieved were that in addition to performing reasonably realistically, critical velocities were observed at lower values in curves compared to straights. Also, the possibility of centrifugal forces acting in a stabilizing fashion were touched upon.

The appendix lists the definitions of variables used throughout the paper, coordinate transformations employed, a more detailed view on the calculation of creepage, a more detailed view on the penetration modelling aspect of the wheel/rail interaction, and last but not least, program code.

Contents

Chapter 1

Introduction

Trains are known to suffer ill dynamical effects at high speed, and a noticeable phenomenon is the hunting-motion experienced by the railcar bogie. This hunting motion is a steady side to side motion of the bogie, that ultimately signals a loss of stability of the formerly stable center position of the railcar bogie on the track.

The significant effects of this phenomenon include but is not limited to increaed rail/wheel wear, potential damage to cargo and passenger discomfort. Ultimately, this all adds to running costs. In the extreme case, these dynamical effects may lead to derailment of the train, and unacceptable losses.

We do not truly understand how this phenomenon occurs intuitively, but by establishing a nonlinear mathematical model of the train, we can accurately reproduce these effects, and thus approach an explanation for what is going on. Especially interesting is to see how the critical velocity at which this phenomenon becomes possible occurs at lower speed in curves. A more detailed look into the dynamics is offered later in this report.

The main aim of this report is to present a mathematical model for the well known railcar bogie due to Cooperrider. This bogie model exhibits many of the dynamical phenomena inherent to many bogies in use today, and yet is simpler to model than other bogies. This model is then to implemented such that we can investigate the behaviour of the bogie on a curved, superelevated track. This, coupled with the rail-wheel contact dynamics determined through the use of the theory of Henrich Hertz and the Shen-Hedricks-Elkins method completes our railcar bogie model.

Following this, we set out to experiment on our model, by modifying three main parameters : bogie velocity, curve radius, and track superelevation, and investigating the aforementioned effect on the associated velocities where the hunting phenomenon becomes possible.

Chapter 2

The Model

In this chapter we will present the model we have investigated. We are interested in the behaviour of a train running through a curve. Since the motion of a train is very affected by the bogie we will concentrate on the behaviour of the bogie. However, since we wish to model a train going trough a curve we have to take into account the centrifugal force that will affect the car body, and hence we will model the bogie and a half car body since we assume two bogies per car body.

We have used the complex bogie model formulated by Cooperrider as a model for the bogie. This model consists of two wheelsets and a bogie frame that are connected by springs and dampers, and the car body is also connected to the bogie frame by springs and the dampers. These springs and dampers are modelled to be linear. We have illustrated our model in figures 2.1 to 2.3.

We consider our bogie model running through a right hand curve as shown in figure 2.4. It is well known that the contact forces between the wheel and rails are very important in an investigation of the dynamics of a train. Therefore, we have used the standard *UIC60* profile to model the rails and the *DSB97-1* to model the wheel profile in order to make the model realistic. The rail profile has an inclination of $\frac{1}{40}$ towards the center of the track. DSB97-1 is much like the european standard profile S1002 (see e.g. [1] for further details).

We refer to table 2.1 for the specific values of the constants, and to appendix A for definitions in general regarding our model.

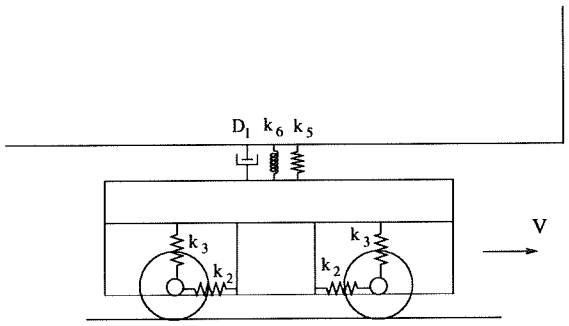


Figure 2.1: Profile view of the model.

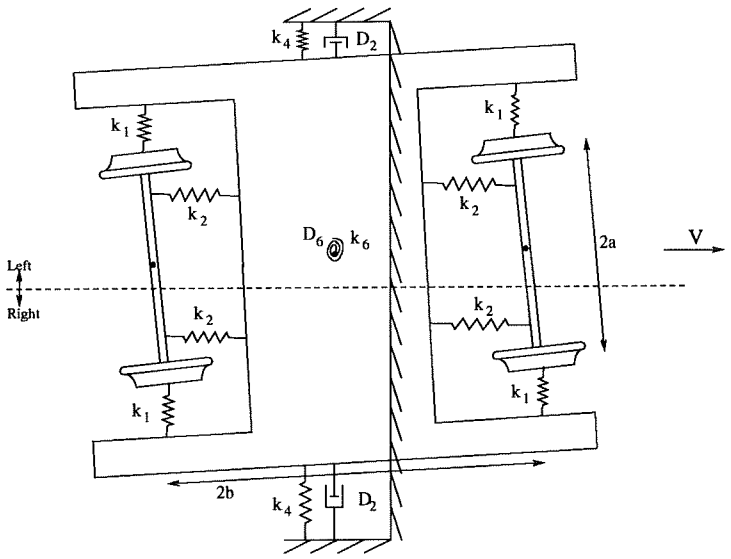


Figure 2.2: Top view of the model.

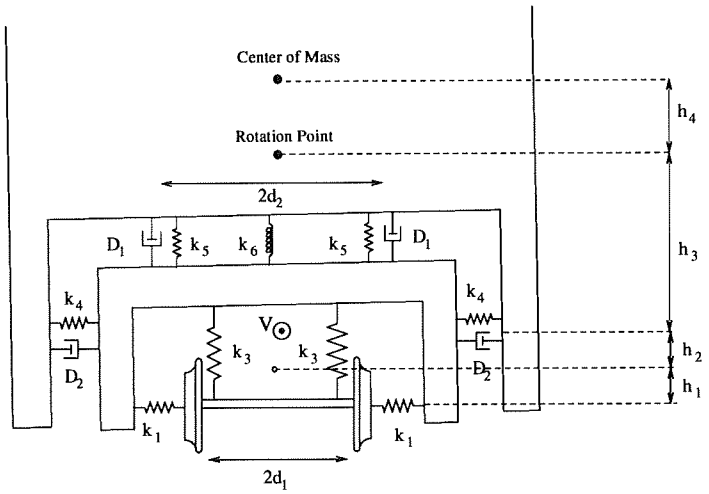


Figure 2.3: Front view of the model.

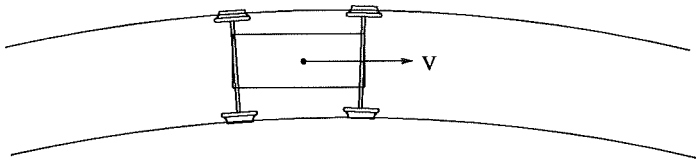


Figure 2.4: The right hand curve under considaration.

$k_1 = 1823.0$ kN/m	$k_2 = 3646.0$ kN/m	$k_3 = 3646.0$ kN/m
$k_4 = 182.3$ kN/m	$k_5 = 333.3$ kN/m	$k_6 = 2710.0$ kNm
$D_1 = 20.0$ kNs/m	$D_2 = 29.2$ kNs/m	$a = 0.75$ m
$b = 1.074$ m	$d_1 = 0.62$ m	$d_2 = 0.68$ m
$h_1 = 0.0762$ m	$h_2 = 0.6584$ m	$h_3 = 0.8654$ m
$h_4 = 0.0346$ m	$m_w = 1022.0$ kg	$m_f = 2918.9$ kg
$m_c = 44388.0$ kg	$I_{wy} = 80.0$ kgm^2	$I_{wx} = 678.0$ kgm^2
$I_{wz} = 678.0$ kgm^2	$I_{fx} = 6780.0$ kgm^2	$I_{fy} = 6780.0$ kgm^2
$I_{fz} = 6780.0$ kgm^2	$I_{cx} = 2.80 \cdot 10^5$ kgm^2	$r_0 = 0.425$ m

Table 2.1: Constants

2.1 Degrees of Freedom

The model has 16 degrees of freedom and are given in table 2.2.

q_1	Front wheelset lateral
q_2	Front wheelset yaw
q_3	Rear wheelset lateral
q_4	Rear wheelset yaw
q_5	Bogie frame lateral
q_6	Bogie frame yaw
q_7	Bogie frame roll
q_8	Rail car roll
q_9	Front wheelset vertical
q_{10}	Rear wheelset vertical
q_{11}	Front wheelset roll
q_{12}	Rear wheelset roll
q_{13}	Bogie frame vertical
q_{14}	Bogie frame pitch
β_f	Angular velocity perturbation for the front wheelset
β_r	Angular velocity perturbation for the rear wheelset

Table 2.2: Degrees of freedom

2.2 Coordinate Systems

We have used three coordinate systems.

- (X_r, Y_r, Z_r)
  A reference frame moving along with the velocity of the bogie. The subscript is for rail. Each body has its own rail coordinate system and the origin is placed in the center of mass of the specific body when it is in centered position. The frame is not an inertial frame of reference, since it follows the bogie through the curve. This gives rise to fictive forces. Positive directions are defined in figure 2.6.

- (X_w, Y_w, Z_w)
  A coordinate system that follows the wheelsets. The origin is placed in the center of mass of the wheelsets. Positive directions are defined in figure 2.7.

- (X_c, Y_c, Z_c)
  A coordinate system that follows the contact plane between the wheel and rail. The origin is placed in the contact point. The contact coordinate system is equal to the wheelset coordinate system rotated with the conicity around its lateral axis and translated to the contact point (see figure 2.8).

The rotation of a body in one of the coordinate systems is given by the three angles ϕ (roll), θ (pitch) and ψ (yaw), which denotes the rotation around the x-axis, y-axis and z-axis, respectively. Positive directions for these angles are found in figure 2.5.

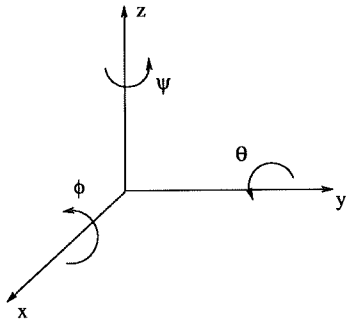


Figure 2.5: Positive directions for the angles (right hand rule).

We switch coordinate system by the following orthogonal transformation matrices (for further details see appendix B). The appended subscript letter l

and r from the contact coordinate system denote left and right wheels, respectively.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} 1 & \psi & 0 \\ -\psi & 1 & \phi \\ 0 & -\phi & 1 \end{bmatrix} \cdot \begin{bmatrix} X_r \\ Y_r \\ Z_r \end{bmatrix}$$

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_l & -\sin\delta_l \\ 0 & \sin\delta_l & \cos\delta_l \end{bmatrix} \cdot \begin{bmatrix} X_{cl} \\ Y_{cl} \\ Z_{cl} \end{bmatrix}$$

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_r & \sin\delta_r \\ 0 & -\sin\delta_r & \cos\delta_r \end{bmatrix} \cdot \begin{bmatrix} X_{cr} \\ Y_{cr} \\ Z_{cr} \end{bmatrix}$$

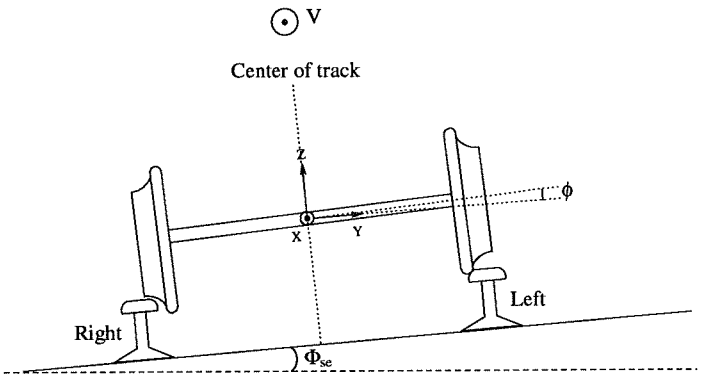


Figure 2.6: Rail coordinate system.

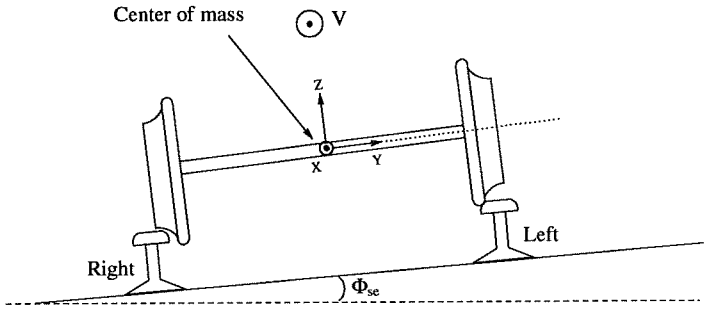


Figure 2.7: Wheelset coordinate system.

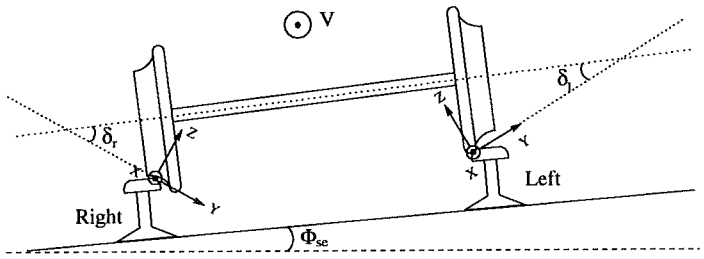


Figure 2.8: Contact coordinate system for the left and right wheel. The x-axis of the contact coordinate system is parallel to the rail only if the yaw angle of the wheelset is zero.

2.3 Differential Equations

We have used Newton's second law in order to make a mathematical description of our model. The result of this is shown below, and hence by analysing a set of ordinary differential equations we can achieve knowledge of the dynamics of the bogie. We have assumed that all state variables are small, which is used to linearize trigonometric functions. To simplify the equations we have used $m_x = \frac{1}{4}m_c + \frac{1}{2}m_f + m_w$ for the static load on each wheelset. The tangential forces in the wheel-rail contact are denoted F and the normal forces with N. Each tangential and normal force has an index e.g. F_{fly}. This means the tangential force for the **f**ront **l**eft wheel in the **y**-direction.

$$
\begin{aligned}
m_w\ddot{q}_1 &= F_{fly} + F_{fry} + N_{fly} + N_{fry} - 2k_1(q_1 - q_5 - bq_6 - h_1q_7) - m_xg\Phi_{se} + \frac{m_wV^2}{R} \\
I_{wz}\ddot{q}_2 &= a_{fr}(F_{frx} + (F_{fry} + N_{fry})q_2) - a_{fl}(F_{flx} + (F_{fly} + N_{fly})q_2) - 2k_2d_1^2(q_2 - q_6) \\
m_w\ddot{q}_3 &= F_{rly} + F_{rry} + N_{rly} + N_{rry} - 2k_1(q_3 - q_5 + bq_6 - h_1q_7) - m_xg\Phi_{se} + \frac{m_wV^2}{R} \\
I_{wz}\ddot{q}_4 &= a_{rr}(F_{rrx} + (F_{rry} + N_{rry})q_4) - a_{rl}(F_{rlx} + (F_{rly} + N_{rly})q_4) - 2k_2d_1^2(q_4 - q_6) \\
m_f\ddot{q}_5 &= 2k_1(q_1 + q_3 - 2q_5 - 2h_1q_7) + 2k_4(h_2q_7 + h_3q_8 - q_5) + 2D_2(h_2\dot{q}_7 + h_3\dot{q}_8 - \dot{q}_5) \\
&\quad -m_fg\Phi_{se} + \frac{m_fV^2}{R} \\
I_{fz}\ddot{q}_6 &= 2d_1^2k_2(q_2 + q_4 - 2q_6) - k_6q_6 - D_6\dot{q}_6 + 2bk_1(q_1 - q_3 - 2bq_6) \\
I_{fx}\ddot{q}_7 &= 2k_1h_1(q_1 + q_3 - 2q_5 - 2h_1q_7) + 2k_4h_2(q_5 - h_2q_7 - h_3q_8) + \\
&\quad 2D_2h_2(\dot{q}_5 - h_2\dot{q}_7 - h_3\dot{q}_8) - 2d_1^2k_3(2q_7 - q_{11} - q_{12}) \\
&\quad -2d_2^2(k_5(q_7 - q_8) + D_1(\dot{q}_7 - \dot{q}_8)) \\
I_{cx}\ddot{q}_8 &= -2d_2^2(k_5(q_8 - q_7) + D_1(\dot{q}_8 - \dot{q}_7)) + 2k_4h_3(q_5 - h2q_7 - h3q_8) + \\
&\quad 2D_2h_3(\dot{q}_5 - h2\dot{q}_7 - h_3\dot{q}_8) + h_4m_cg(\Phi_{se} + q_8) - h_4\frac{m_cV^2}{R} \\
m_w\ddot{q}_9 &= F_{flz} + F_{frz} + N_{flz} + N_{frz} + 2k_3(q_{13} - q_9) - m_xg - \frac{m_wV^2}{R}\Phi_{se} \\
m_w\ddot{q}_{10} &= F_{rlz} + F_{rrz} + N_{rlz} + N_{rrz} + 2k_3(q_{13} - q_{10}) - m_xg - \frac{m_wV^2}{R}\Phi_{se} \\
I_{wx}\ddot{q}_{11} &= -a_{fr}(F_{frz} + N_{frz} - (F_{fry} + N_{fry})q_{11}) + a_{fl}(F_{flz} + N_{flz} - (F_{fly} + N_{fly})q_{11}) \\
&\quad -2k_3d_1^2(q_{11} - q_7) \\
I_{wx}\ddot{q}_{12} &= -a_{rr}(F_{rrz} + N_{rrz} - (F_{rry} + N_{rry})q_{12}) + a_{rl}(F_{rlz} + N_{rlz} - (F_{rly} + N_{rly})q_{12}) \\
&\quad -2k_3d_1^2(q_{12} - q_7) \\
m_f\ddot{q}_{13} &= -2k_3(2q_{13} - q_9 - q_{10}) - 2k_5q_{13} - 2D_1\dot{q}_{13} - \frac{m_fV^2}{R}\Phi_{se} \\
I_{fy}\ddot{q}_{14} &= -2bk_3(2bq_{14} + q_9 - q_{10}) \\
I_{wy}\dot{\beta}_f &= -r_{fr}(F_{frx} + (F_{fry} + N_{fry})q_2) - r_{fl}(F_{flx} + (F_{fly} + N_{fly})q_2) - 2d_1^2k_3q_2q_7 \\
I_{wy}\dot{\beta}_r &= -r_{rr}(F_{rrx} + (F_{rry} + N_{rry})q_4) - r_{rl}(F_{rlx} + (F_{rly} + N_{rly})q_4) - 2d_1^2k_3q_4q_7
\end{aligned}
$$

The distance between the center of mass of the wheelset and the contact point on a specific wheel (a_{fl}, a_{fr}, a_{rl} and a_{rr}) and the actual rolling radius (r_{fl}, r_{fr}, r_{rl} and r_{rr}) are provided by the program *RSGEO*. This program finds the point of contact given any lateral displacement of the wheelset.

2.4 Creep Forces

In this section we will find expressions for the tangential forces or creep forces that arise in the wheel-rail contact. As mentioned before, the wheel-rail interaction is very important when analysing the behaviour of a bogie, so we have to take into account the nonlinearities that exists.

Several theories has been developed to approximate the creep forces, and the one we have used is due to Shen, Hedrick and Elkins (SHE). The theory combines Kalker's linear theory with the theory presented by Johnson and Vermuelen, and the result is a nonlinear relationship between the creepforces and the normal forces.

To be able to use SHE we have to find the relative motion between the wheel and rail. This relative motion is called the creepage. This requires some mathematical manipulations and can be found in appendix C. We have used SHE because it is not too difficult to implement, and the approximation for the creep forces is quite good compared to real measurements. We have found the following creep terms.

$$
\begin{aligned}
\xi_{flx} &= \frac{V + (\psi + \frac{b}{R})\dot{y} + r_l((\psi + \frac{b}{R})\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se}}{V} \\
\xi_{frx} &= \frac{V + (\psi + \frac{b}{R})\dot{y} + r_r((\psi + \frac{b}{R})\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se}}{V} \\
\xi_{fly} &= \frac{(-V(\psi + \frac{b}{R})(1 + \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_l)\cos\delta_l + (-\phi\dot{y} + \dot{z} + a_l\dot{\phi})\sin\delta_l}{V} \\
\xi_{fry} &= \frac{(-V(\psi + \frac{b}{R})(1 - \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_r)\cos\delta_r - (-\phi\dot{y} + \dot{z} - a_r\dot{\phi})\sin\delta_r}{V}
\end{aligned}
$$

$$
\begin{aligned}
\xi_{rlx} &= \frac{V + (\psi - \frac{b}{R})\dot{y} + r_l((\psi - \frac{b}{R})\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se}}{V} \\
\xi_{rrx} &= \frac{V + (\psi - \frac{b}{R})\dot{y} + r_r((\psi - \frac{b}{R})\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se}}{V} \\
\xi_{rly} &= \frac{(-V(\psi - \frac{b}{R})(1 + \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_l)\cos\delta_l + (-\phi\dot{y} + \dot{z} + a_l\dot{\phi})\sin\delta_l}{V} \\
\xi_{rry} &= \frac{(-V(\psi - \frac{b}{R})(1 - \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_r)\cos\delta_r - (-\phi\dot{y} + \dot{z} - a_r\dot{\phi})\sin\delta_r}{V}
\end{aligned}
$$

$$\xi_{fls} = \frac{-(\Omega - (\psi + \frac{b}{R})\dot{\phi})\sin\delta_l + \dot{\psi}\cos\delta_l}{V}$$
$$\xi_{frs} = \frac{(\Omega - (\psi + \frac{b}{R})\dot{\phi})\sin\delta_r + \dot{\psi}\cos\delta_r}{V}$$
$$\xi_{rls} = \frac{-(\Omega - (\psi - \frac{b}{R})\dot{\phi})\sin\delta_l + \dot{\psi}\cos\delta_l}{V}$$
$$\xi_{rrs} = \frac{(\Omega - (\psi - \frac{b}{R})\dot{\phi})\sin\delta_r + \dot{\psi}\cos\delta_r}{V}$$

Kalker's linear theory gives the following creep force components

$$\tilde{F}_x = -a_e b_e G C_{11}\xi_x$$
$$\tilde{F}_y = -a_e b_e G\left(C_{22}\xi_y + \sqrt{a_e b_e}C_{23}\xi_s\right)$$

and the resulting creep force is then

$$\tilde{\mathbf{F}}_\tau = \tilde{F}_x\mathbf{e}_x + \tilde{F}_y\mathbf{e}_y$$

where G[1] is the shear modulus and C_{11}, C_{22} and C_{23} are Kalker's creepage coefficients. These coefficients depend on the ratio a_e/b_e and are tabulated by the RSGEO program. We adjust the creepforce from Kalker's linear theory and the result are the creep forces F_x and F_y, and they are given by

$$|\mathbf{F}_\tau| = \begin{cases} \mu N\left(\left[\frac{|\tilde{\mathbf{F}}_\tau|}{\mu N}\right] - \frac{1}{3}\left[\frac{|\tilde{\mathbf{F}}_\tau|}{\mu N}\right]^2 + \frac{1}{27}\left[\frac{|\tilde{\mathbf{F}}_\tau|}{\mu N}\right]^3\right) & \frac{|\tilde{\mathbf{F}}_\tau|}{\mu N} < 3 \\ \mu N & \frac{|\tilde{\mathbf{F}}_\tau|}{\mu N} \geq 3 \end{cases} \tag{2.1}$$

$$\epsilon = \frac{|\mathbf{F}_\tau|}{|\tilde{\mathbf{F}}_\tau|}$$
$$F_x = \epsilon\tilde{F}_x \qquad F_y = \epsilon\tilde{F}_y$$

The friction coefficient μ is chosen to be 0.15 throughout our experiments. The creep-creepforce relationship given in equation (2.1) is in general not realistic since the creepforce will decay when the wheels are spinning, but since we do not have any torque on the wheel axles we use the relationship above.

2.5 Normal Forces

The normal forces generated at the contact points arise directly from Newton's third law : every action force has an equal and opposite reaction force.

[1] $G = \frac{21\cdot10^{10}}{2\cdot(1+0.27)}\frac{N}{m^2} \approx 8.27\cdot10^{10}\frac{N}{m^2}$

We also consider the wheel and rail to be two elastic bodies, and are thus subject to deformation. In order to model the actual deformation, we consider the two bodies to penetrate into one another without deforming, and use the fact that the normal force depends on this fictitious penetration.

In our effort to determine the normal forces, we are then required to somehow determine characteristics of the contact patch, most especially penetration. Ultimately, to determine the normal force, we take advantage of a relationship between wheel-rail penetration and the normal forces generated.

Our first step is the use of theory presented by Henrich Hertz in the late 19^{th} century (1882) in order to determine contact patch characteristics. This enables us to begin to determine the forces by letting us know the dimensions of the elliptic contact patch between the wheel and rail. This, however, comes under the price of the following four conditions:

- The two bodies in contact must be described by a bilinear polynomial in the point of contact.
- The two bodies are made from completely elastic, homogeneous and isotropic materials.
- The displacement in the point of contact can be neglected.
- The diameter of the contact patch is small compared to the characteristic diameters of the two bodies.

By [1], the relationship between contact patch ellipse semi-axes and penetration with respect to the normal force generated is given by:

$$a_e \propto N^{\frac{1}{3}} \quad b_e \propto N^{\frac{1}{3}} \quad N \propto q^{\frac{3}{2}}$$

RSGEO (see section 3.4) will yield the *static* normal force as a function of the lateral displacement of the wheelsets. We then adjust the normal force and the contact ellipse geometries dynamically, because the rolling motion and the vertical displacements of the wheelsets will affect the normal force. We do this by using that the above proportionalities yield the following formula for updating the normal force:

$$N = N_0 \left[1 + \frac{\Delta q}{q_0}\right]^{\frac{3}{2}}$$

where Δq is the additional penetration (see appendix D) given with the help of the lateral displacement of the wheelset (y) and the vertical displacement of the wheelset (z):

$$\begin{aligned}
\Delta q_l &= -(a_{Rl} - y - a_l - \phi r_l)\sin(\delta_l + \phi) + (-z - \phi a_l)\cos(\delta_l + \phi) \\
\Delta q_r &= (-a_{Rr} - y + a_r - \phi r_r)\sin(\delta_r - \phi) + (-z + \phi a_r)\cos(\delta_r - \phi)
\end{aligned}$$

Chapter 3

Numerical Methods

3.1 Implementation of the Model

We chose to implement our model and the numerical treatment of it in the language of C++ since the solver we use (SDIRK) is a package provided exclusively for that language. Visualization was achieved by importing our results into MATLAB and using the powerful visualization tools MATLAB has to produce graphs.

Furthermore, in that SDIRK only treats systems of first order ODEs, our system of 14 second order ODEs and two first order ODEs are transformed into a system of 30 first order ODEs.

The source code can be viewed in appendix E and downloaded from the WWW at `http://www.student.dtu.dk/~c973500/curve/`.

The general structure of our program can be seen in figure 3.1.

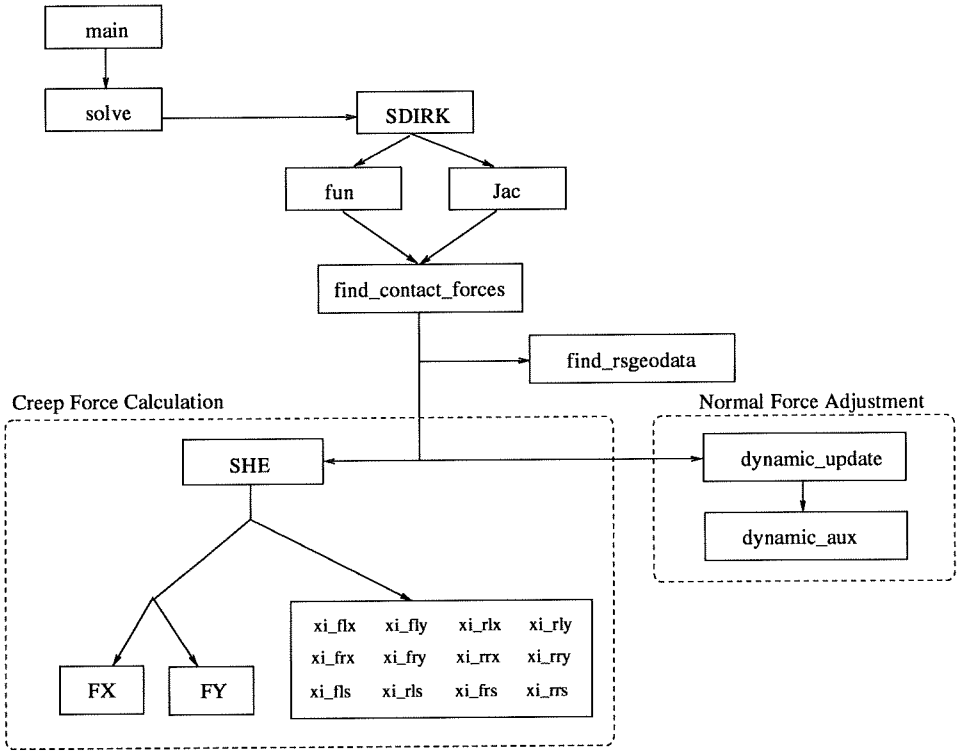

main
solve
SDIRK
fun
Jac
find_contact_forces
find_rsgeodata
Creep Force Calculation
Normal Force Adjustment
SHE
dynamic_update
dynamic_aux
FX
FY
xi_flx xi_fly xi_rlx xi_rly
xi_frx xi_fry xi_rrx xi_rry
xi_fls xi_rls xi_frs xi_rrs

Figure 3.1:

3.2 Program Overview

The entire program is consolidated in one file, namely `model.cc`. The core functions implemented in the file are listed as follows:

1. `void fun`
   This implements the right hand side of our system of first order ODEs, and is central in the utilization of the SDIRK solver.
2. `void jac`
   This calculates the jacobi matrix of our system of first order ODEs and is also central in the operation of the SDIRK solver. This is done in an analytical fashion, as well as an approximative numerical fashion.
3. `double rhs`
   This returns the right hand side value of a particular equation or our system.
4. `void find_contact_forces`
   This method calculates the forces involved generated by the wheel-rail contact.
5. `void fast_num`
   This method calculates precisely the needed jacobi elements which are to be approximated.
6. `void jac_numerical`
   Approximates the derivative by a central difference approximation.
7. `void read_rsgeo`
   This reads in the data from RSGEO into memory.
8. `void find_rsgeodata`
   This uses the RSGEO data and interpolates linearly in order to get precise data depending on the actual lateral wheelset displacement.
9. `void dynamic_update`
   This updates the values given by RSGEO in order to take into account the vertical degrees of freedom.
10. `void dynamic_aux`
    Auxiliary function called by `dynamic_update` in order to update data for a particular wheel.
11. `void SHE`
    This implements the Shen-Hedricks-Elkins method of working out creep forces involved in the wheel-rail contact.
12. `double xi_flx`
    Front left wheel creep in the x-direction (longitudinal).
13. `double xi_frx`
    Front right wheel creep in the x-direction (longitudinal).
14. `double xi_fly`
    Front left wheel creep in the y-direction (lateral).
15. `double xi_fry`
    Front right wheel creep in the y-direction (lateral).

16. `double xi_fls`
    Front left wheel spin creep.
17. `double xi_frs`
    Front right wheel spin creep.
18. `double xi_rlx`
    Rear left wheel creep in the x-direction (longitudinal).
19. `double xi_rrx`
    Rear right wheel creep in the x-direction (longitudinal).
20. `double xi_rly`
    Rear left wheel creep in the y-direction (lateral).
21. `double xi_rry`
    Rear right wheel creep in the y-direction (lateral).
22. `double xi_rls`
    Rear left wheel spin creep.
23. `double xi_rrs`
    Rear right wheel spin creep.
24. `double Fx`
    Creep force with respect to the contact coordinate system in the x-direction (longitudinal).
25. `double Fy`
    Creep force with respect to the contact coordinate system in the y-direction (lateral).
26. `void toFile`
    Outputs the current solution point of our model and contact forces to a file.
27. `void solve`
    Initializes SDIRK, sets up our system with initial conditions and proceeds to solve it. Furthermore, parameters set in `main` can here be modified to continuously change (linearly or otherwise) with time.
28. `void main`
    Sets main parameters velocity, curve radius and superelevation, and then calls `solve`.

Other functions serve as auxiliary functions needed to determine initial conditions for certain experiments, as well as auxiliary output functions for obtaining results and recalling the last output data to be used as initial conditions for a new test run.

In order to run a simulation, one needs only to modify few lines of code in `main` and `solve` in order to set up the conditions as needed. The main paramaters velocity, curve radius and superelevation are all set in `main`, while things governing the timespan for SDIRK and other related SDIRK parameters (accuracy, stepsize, etc.) are modified in `solve`. Furthermore, it is here one can specify the linear alteration of the main parameters with time.

Initial conditions are set just prior to calling SDIRK to begin solving. To do this, the user has the choice of explicitly entering values, or by calling an already defined function that sets the initial conditions such that the desirable steady state is achieved for some variation of the parameters. These auxiliary functions can be seen in the code for `model.cc` on the homepage, and have been omitted in the appendix.

Finally, in order to compile the code, a makefile has been included in the appendix that performs this job by calling the command `make`[1]. Furthermore, by appending the option "`clean`" when using the command `make`, all old data files and emacs[2] backup files are removed. The makefile code can be found in appendix E.2.

3.3 SDIRK

In the treatment of our problem, we employed the numerical solver SDIRK, as provided by prof. Per Grove Thomsen[3]. This solver was employed by [5] and less so by [1].

This solver belongs to the Runge-Kutta family of solvers, and more precisely, the singly diagonally implicit Runge-Kutta solvers (which is the origin of the acronym SDIRK). As such, it is an efficient solver for stiff problems, yet requires the specification of the Jacobi matrix of our system, as well as the right hand side of our system of first order ODEs.

In having to deliver the Jacobi matrix of our system, which has 900 elements (30×30 matrix), we are able to deliver many entries analytically, however, the ODEs which depend on the rail-wheel contact must be calculated numerically. This is done by perturbing the solution slightly, recalculating values, and then determining the difference. This pertubation is done to both sides of the solution, in order to yield a better estimate, and can be seen in the function `jac_numerical` in the source code of our main model file, `model.cc` in appendix E.1.

With respect to error tolerances and timesteps, we utilized an error tolerance of 10^{-8} and an output stepsize of $h = 0.001$.

[1] This command is only available under UNIX based systems, and Windows systems which have UNIX environments installed, e.g. Cygwin.

[2] A text editing utility.

[3] Institute for Informatics and Mathematical Modelling, Technical University of Denmark.

Column	Description
1	Lateral displacement of the wheelset (m).
2	Normal force in the contact point coordinate system (N).
3	Angle between the wheel axle and the contact plane (rad).
4	Semi-axis of the contact ellipse in the longitudinal direction (m).
5	Semi-axis of the contact ellipse in the lateral direction (m).
6	Lateral distance to the contact point (positive) (m).
7	Actual rolling radius (positive) (m).
8	Kalker's creepage coefficient C_{11}.
9	Kalker's creepage coefficient C_{22}.
10	Kalker's creepage coefficient C_{23}.
11	Vertical position of the contact point on the rail measured from the top of the rail (m).
12	Static penetration depth (m).
13	Lateral distance to the contact point on the rail in the rail coordinate system (m).

Table 3.1: Datastructure of the table produced by RSGEO.

3.4 RSGEO

The program RSGEO is not actually available to us, but whenever referred to, we generally refer to the table of results that RSGEO calculates that enables us to calculate the contact patch geometries between the wheel and rail. This data was obtained from [1] and used directly. This data is given directly as a function of the wheelset displacement, and nothing more. Ultimately, it might be desirable to have all this data also as a function of wheelset yaw, but these angles tend to be quite small, and thus arguably have neglible effects with regards to some of the data.

The data of the RSGEO file is arranged in column-wise fashion, with 13 columns representing the data laid out in table 3.1.

This data is initially read into memory in our implementation, and constantly referred to every timestep in order to determine contact patch characteristics, and ultimately determine the status of the wheels, contact points and forces involved.

Chapter 4

Vehicle Dynamics

It has been well known among engineers for a long while that railway vehicles above a certain speed exhibit a sideways oscillation on the track. This lateral movement is called the hunting motion and the characteristic speed for the onset of the hunting motion is known as the critical velocity. In general, this depends on many factors, e.g. track geometry, wheel profile, coefficient of friction and design of the railcar bogie.

The critical velocity comes in two versions, a linear critical velocity and a nonlinear critical velocity, and to understand the difference between these two velocities one should have the bifurcation diagram given in figure 4.1 in mind. From the figure we see that the linear critical velocity is defined at the point where the system undergoes a subcritical Hopf bifurcation. The nonlinear critical velocity exists at the saddle node bifurcation. It is important to notice that the centered position in the track is a stable solution until the bogie reaches the linear critical velocity.

Of special consequence, for velocities larger than the nonlinear critical velocity there exists a stable solution (the hunting motion) with an amplitude different from zero. In practice, the bogie is affected by irregularities in the track and other disturbances, so it is most likely that the bogie begins to hunt at lower velocities than the linear critical velocity. Because of this, the nonlinear critical velocity is often just referred to as the critical velocity.

The effects of the hunting motion, among others, are increaed rail/wheel wear and passenger discomfort, and therefore one wishes to avoid this motion. One could think that the critical velocity for a vehicle is a measurable property once and for all by the manufacturer, and then the driver just has to make sure that the train runs below this speed. In reality, this is not the case. The reason for this is that the critical velocity depends on many factors, even for the same vehicle. Changes in the coefficient of friction, the torque on the driven axles or the negotiation of a curve can have a big influence on

the critical velocity. In this report we will focus on the dynamics of a bogie going through a curve.

We do not have a general theory for calculating critical velocities, however, there are some general guidelines that tells us what will affect the critical velocity. For instance, it is well known that the difference in rolling radii, making an effective conicity, in general has a destabilizing effect, meaning that the critical velocity is lower for larger values of conicity. For our special interest in curving dynamics we expect that the critical velocity is lower in the curve, because the effective conicity is larger. The reason for this is that the centrifugal force will create an equilibrium solution which is not in the center of the track, and hence the effective conicity is most likely greater than in the track-centered position.

The question is whether both the linear and nonlinear critical velocities are affected equally or how they might behave differently. One possibility is that the linear and nonlinear critical velocities merge, making the subcritical and saddle node bifurcation a supercritical Hopf bifurcation as illustrated in figure 4.2. The big difference between these two cases is that there is no dangerous subcritical Hopf bifurcation or hysteresis effect.

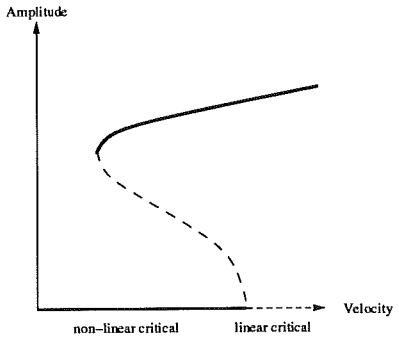


Figure 4.1: Bifurcation diagram illustrating the difference between the linear and non-linear velocity

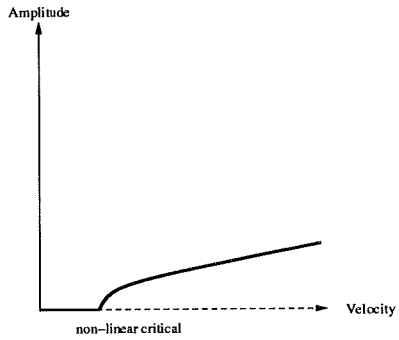


Figure 4.2: Bifurcation diagram illustrating the supercritical Hopf bifurcation.

Chapter 5

Results

5.1 Straight Track

In order to ascertain some idea of how our model stands up to general knowledge, we ran it on straight tracks before subjecting it to curves. This allowed us to compare general behaviour of the model with reality, as well as comparing the critical velocities we obtained with e.g. [1] and [5].

5.1.1 General Behaviour

We first present some general figures of how the bogie behaves when running on the straight track. Figure 5.1 illustrates how the bogie running at 40 m/s will tend to come to rest in a centered position on the track. Provided we are below the critical velocities, which we are, this is expected. Figure 5.2 coupled with the previous figure illustrate how the train 'weaves' its way along the track.

We expect that as we proceed beyond the linear critical velocity for the bogie, the centered position becomes an unstable solution, and the bogie will begin a hunting motion. This is verified in figures 5.3 and 5.4, where we have a bogie running at 130 m/s.

What is interesting to note is that the bogie frame is displaced more than the wheelset, but that is expected due to the fact that the wheels are the ones in contact with the rails, and that they are equipped with flanges. Also of interest is how the yaw motion of the frame is delayed with respect to the wheelsets, but that is also expected, since it is the frame that is subjected to the motion of the wheelsets.

Figures 5.5 and 5.6 illustrate a frequency analysis on the front wheelset's lateral and yaw motion, respectively. The power spectra illustrate that the

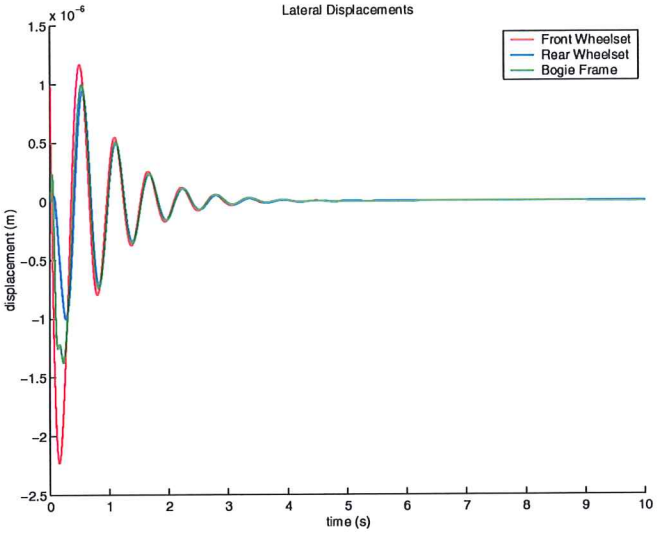


Figure 5.1: Lateral displacement of the bogie running on a straight track. The speed is 40 $\frac{m}{s}$. We see that the wheelsets and bogie frame approaches their centered position as expected.

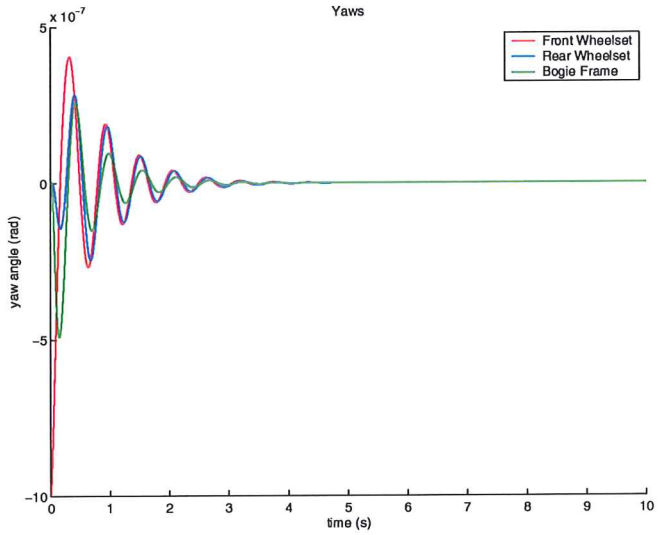


Figure 5.2: Yaw angle of the bogie running on a straight track. The speed is 40 $\frac{m}{s}$. The yaw angle of the bogie frame and wheelsets approaches zero.

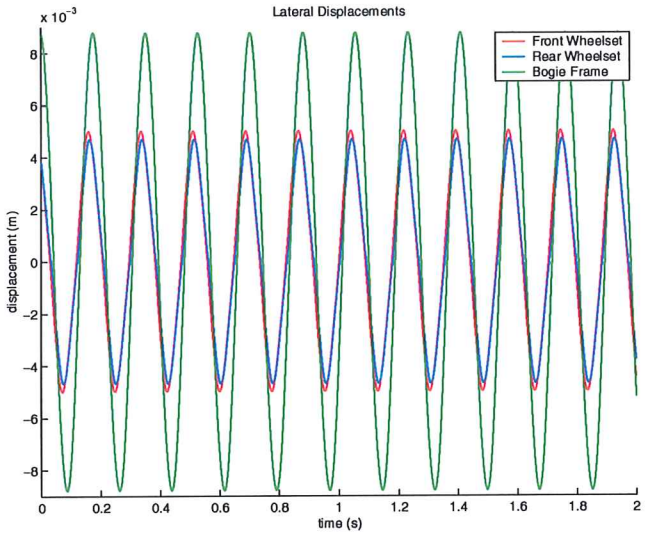


Figure 5.3: Lateral displacement of the bogie running on a straight track. The speed is 130 $\frac{m}{s}$. We see that the velocity has reached a magnitude that makes the zero solution unstable. The bogie is hunting.

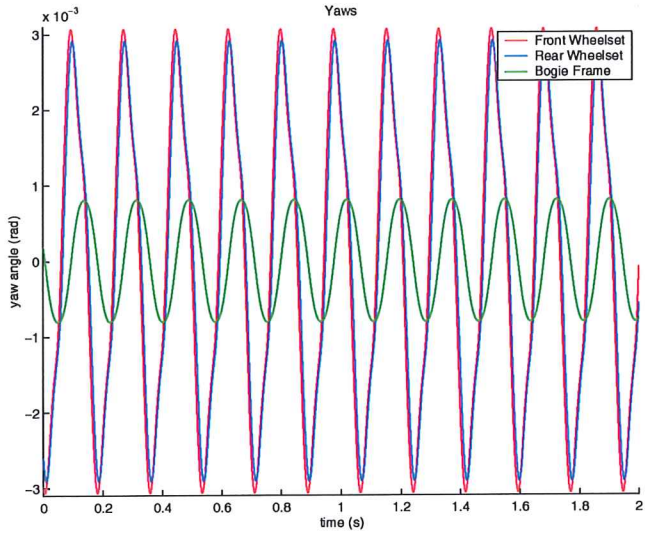


Figure 5.4: Yaw angle of the bogie running on a straight track. The velocity is 130 $\frac{m}{s}$. The hunting motion is also seen in this figure.

lateral and yaw motion of the wheelset is phaselocked to the same frequency of just under 6 Hz.

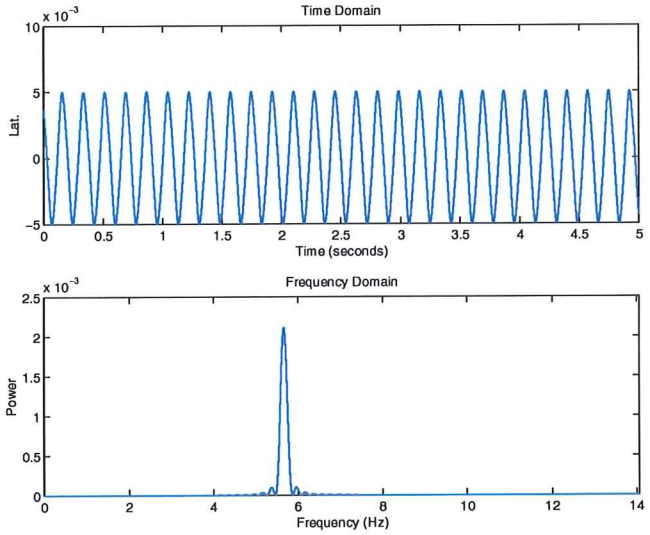


Figure 5.5: Time series and power spectrum for the lateral displacement of the front wheelset. The velocity is 130 $\frac{m}{s}$.

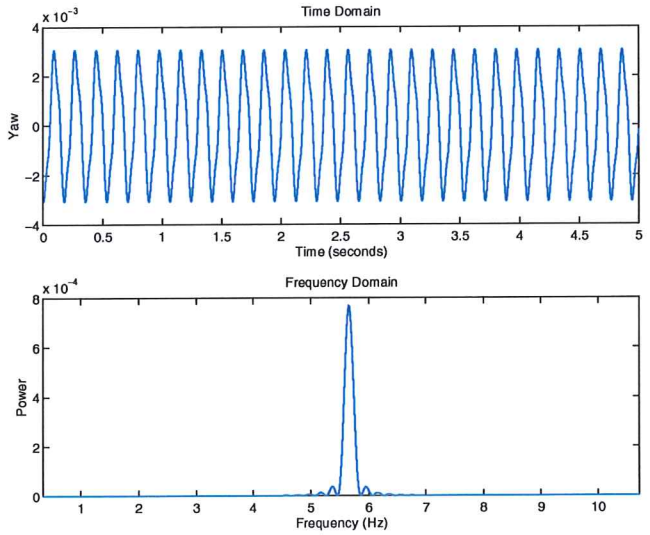


Figure 5.6: Time series and power spectrum for the yaw angle of the front wheelset. The velocity is 130 $\frac{m}{s}$.

Figures 5.7 and 5.8 are phaseplots of the lateral displacement and yaw

angle of the front and rear wheelsets, respectively. This is again for the bogie running above the linear critical velocity at a speed of 130 m/s. The figures should be 'read' in an clockwise fashion, and this gives a good indication of how the train moves on the track.

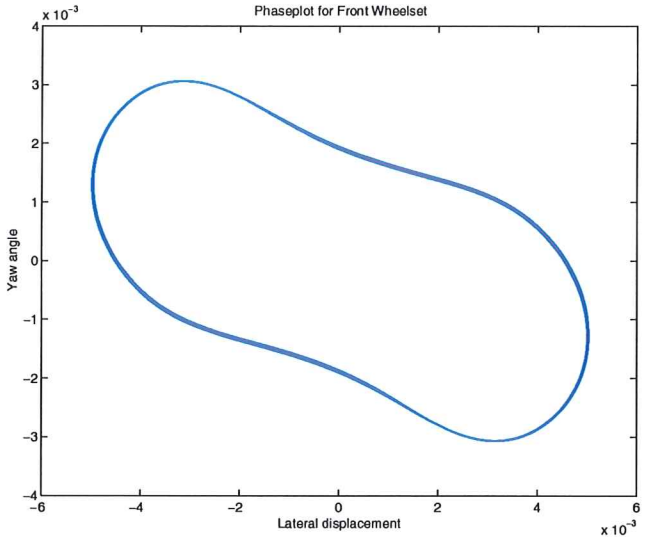


Figure 5.7: Phaseplot illustrating the lateral displacement and the yaw angle of the front wheelset. The velocity is 130 $\frac{m}{s}$.

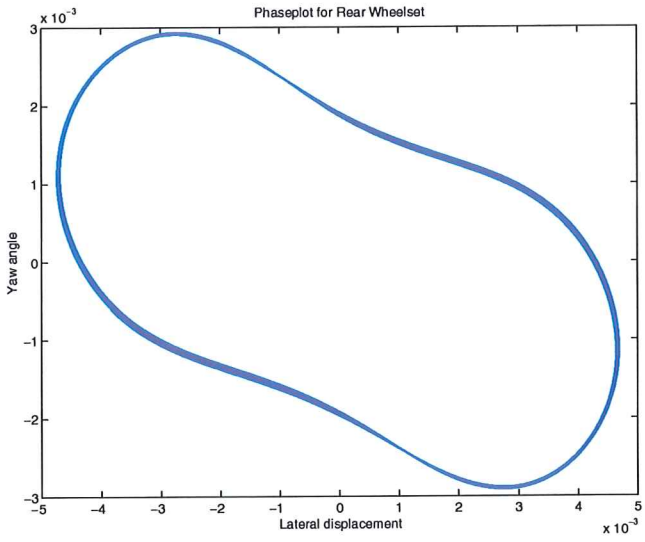


Figure 5.8: Phaseplot illustrating the lateral displacement and the yaw angle of the rear wheelset. The velocity is 130 $\frac{m}{s}$.

5.1.2 Linear Critical Velocity

In further analyzing the validity of our bogie model, we searched for the linear critical velocity on a straight track. Figure 5.9 illustrates the migration of eigenvalues at speeds near the linear critical velocity. This was done by incrementing the velocity of the bogie by 0.1 m/s over a short range of velocities where the bifurcation was suspected to occur.

Figure 5.10 shows a more detailed view of the migration, and highlights the complex conjugate eigenvalues crossing the imaginary axis simultaneously at $v \approx 117.3$ m/s. It is this event that leads to a subcritical Hopf bifurcation, and the subsequent hunting motion of the bogie. The Hopf bifurcation is subcritical, since no slow growth of the attractor occurs, instead, the solution at the formerly stable center position will 'explode' out on to a distant attractor.

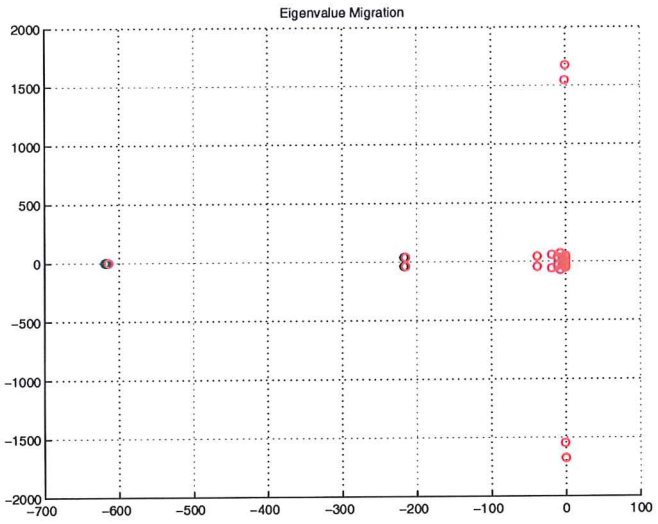


Figure 5.9: Eigenvalue migration near the subcritical Hopf bifurcation.

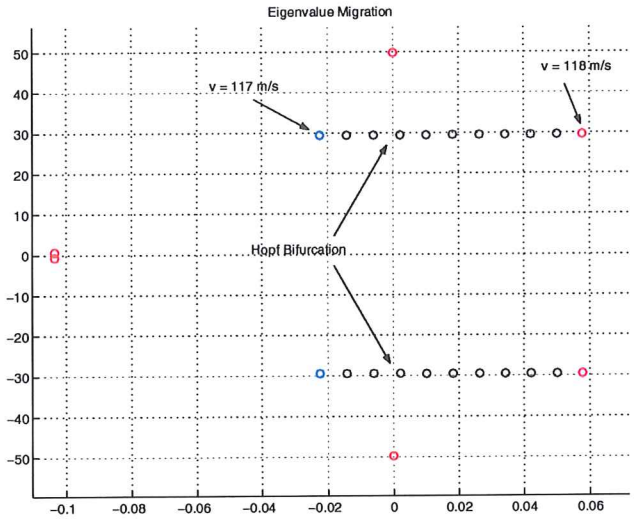


Figure 5.10: Zoomed view of the eigenvalue migration. The Hopf bifurcation takes place when the two complex conjugate eigenvalues crosses the imaginary axis. The linear critical velocity is determined to 117.3 $\frac{m}{s}$.

5.1.3 Nonlinear Critical Velocity

Also of interest in verifying the results of our model is the nonlinear critical velocity. In order to detect this value, we follow the distant attractor corresponding to the hunting motion by beginning at high speed and making sure that we are hunting. When this is done, we slowly reduce speed adiabatically, and when we reduce speed to under the nonlinear critical velocity, the only stable solution for our bogie should be the center position of the track.

Figure 5.11 shows how we've proceeded according to the aforementioned method, and actually found that the bogie stopped hunting once we slowed down enough. Figure 5.12 gives a more detailed view of where the saddle node bifurcation is. The nonlinear critical velocity was ascertained to lie at $v \approx 53.97$ m/s.

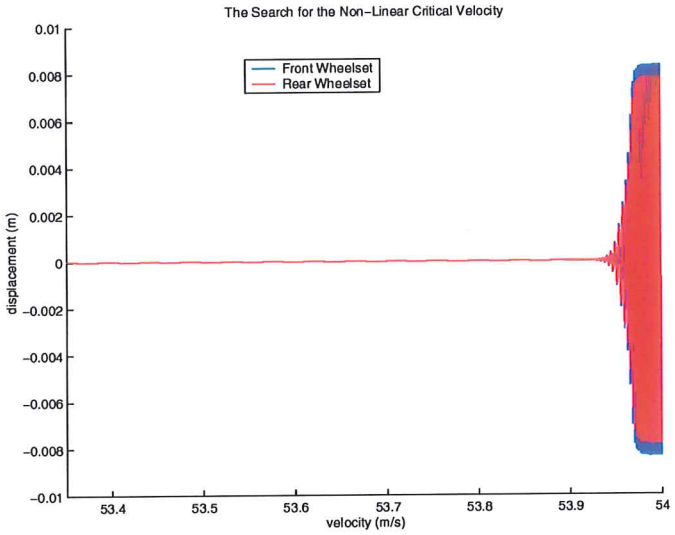


Figure 5.11: The saddlenode bifurcation is found by an adiabatic reduction of the velocity.

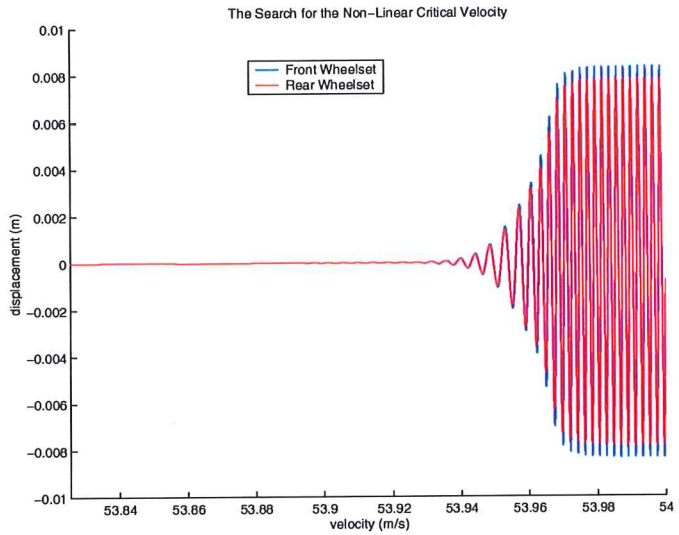


Figure 5.12: Zoomed view of the saddlenode bifurcation. We determine the nonlinear critical velocity to 53.97 $\frac{m}{s}$.

5.2 Curved Track

Having more or less verified our model on a straight track, we move on to examine its behaviour in a curve. The fundamental difference with respect to earlier tests on straight tracks, is that the former equilibrium at the center position of the track is shifted due to the centrifugal and gravitational forces experienced by the train in the curve.

5.2.1 General Behaviour

Figure 5.13 and 5.14 display the asymptotic lateral displacement and yaw angle of the bogie as a function of the curve radius. These measurements were obtained for a velocity of 30 m/s and curve superelevation was set to 2°. This clearly validates our previous statement about the shift of the equilibrium. The dashed lines show the theoretical ideal yaw angle for the rear and front wheelsets, if they were to negotiate the curve as parallel as possible. Figure 5.15 and 5.16 display the same for 50 m/s.

Figure 5.17, 5.18, 5.19 and 5.20 show the same as previous figures but for a superelevation of 4°.

From these figures, we can clearly see that the front wheelset has a tendency to seek outwards in a stronger fashion than the rear wheelset. We see that for 2° superelevation, the equilibrium tends to be further displaced from the center of the track at higher velocities for moderate to large curve radii. For low curve radii, there is barely any difference between wheelset equilibria for $v = 30$ or 50 m/s. The bogie frame, though, does shift outwards for all curve radii, but this is also expected since it is isolated from the wheel/rail contact through springs and dampers, which ultimately allows it a greater range of motion than the flange equipped wheelsets.

We also see that when increasing superelevation, and maintaining the same speed $v = 30$ m/s, we see that the bogie is still displaced outward, but less so, since gravitational forces pulling the train laterally towards the center of the curve are stronger. This is also the case for $v = 50$ m/s.

With respect to yaw, we see that for a superelevation of 2° and $v = 30$ m/s that the yaw angle for the front wheelset has a tendency to not turn sufficiently along with the curve, leading the front wheelset outwards. This effect is, however, removed when the velocity of the bogie is set to $v = 50$ m/s, and in this case the yaw angle matches the theoretical curve quite well. This seems to lead to the conclusion that 2° superelevation is too great for the lower speed of $v = 30$ m/s, and more suitable for $v = 50$ m/s.

For a superelevation of 4° and $v = 30$ m/s there is a poor matching between the ideal values and the results from our model, but for a velocity

of $v = 50$ m/s this difference becomes smaller. Again, this is a poor choice of superelevation for lower velocities, but now it seems that even $v = 50$ m/s is slow.

The reason for why we do not investigate curve radii greater than 1200 m for $v = 50$ m/s is because we end up above the linear critical velocity, and no longer achieve a stable non-oscillating run on the tracks. This is discussed in the following.

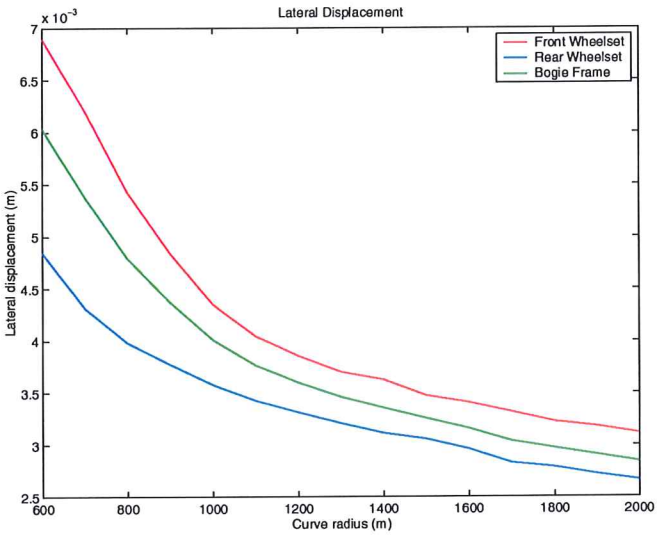


Figure 5.13: Lateral displacement of the bogie as a function of the curve radius. The speed is 30 $\frac{m}{s}$ and the superelevation is 2°.

5.2.2 Critical Velocities

In a further effort to examine the dynamics of our bogie in a curve, we searched for the critical velocities as a function of curve radius. Figure 5.21 presents our findings. As we see, for curves in general, the critical velocities are lower than on a straight track.

What is interesting to note is that the tendency for the critical velocities to decrease does not happen monotonously with the lowering of the curve radius. We do see that at one point, the velocities rise rather sharply, only to decrease monotonously once we continue lowering the curve radius. This was an unexpected observation, but may be explained by the fact that the centrifugal force may begin to act in a stabilizing fashion.

We also see that the nonlinear and linear critical velocities seem to merge at some places, and this could be the effect of solely the linear critical velocity

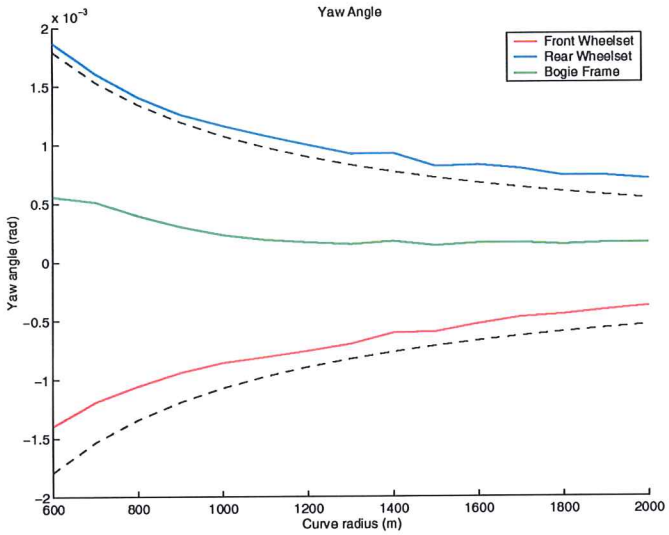


Figure 5.14: Yaw angle of the bogie as a function of the curve radius. The speed is 30 $\frac{m}{s}$ and the superelevation is 2°. The dashed lines are the ideal yaw angle of the wheelsets for which the wear is minimum.

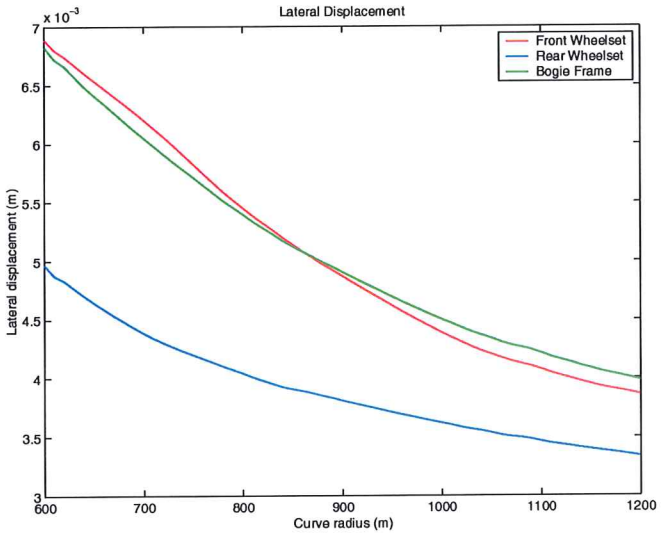


Figure 5.15: Lateral displacement of the bogie as a function of the curve radius. The speed is 50 $\frac{m}{s}$ and the superelevation is 2°.

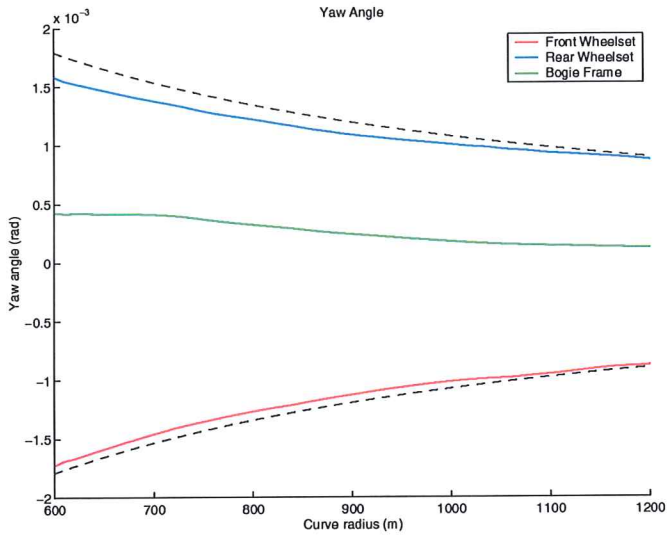


Figure 5.16: Yaw angle of the bogie as a function of the curve radius. The speed is 50 $\frac{m}{s}$ and the superelevation is 2°. The dashed lines are the ideal yaw angle of the wheelsets for which the wear is minimum.

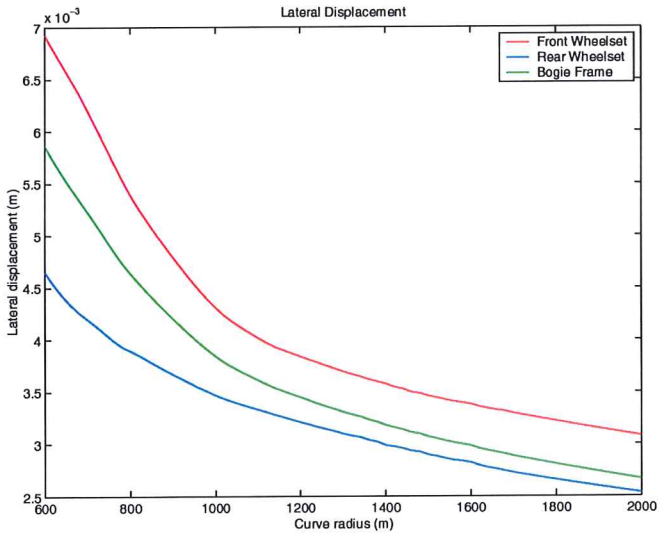


Figure 5.17: Lateral displacement of the bogie as a function of the curve radius. The speed is 30 $\frac{m}{s}$ and the superelevation is 4°.

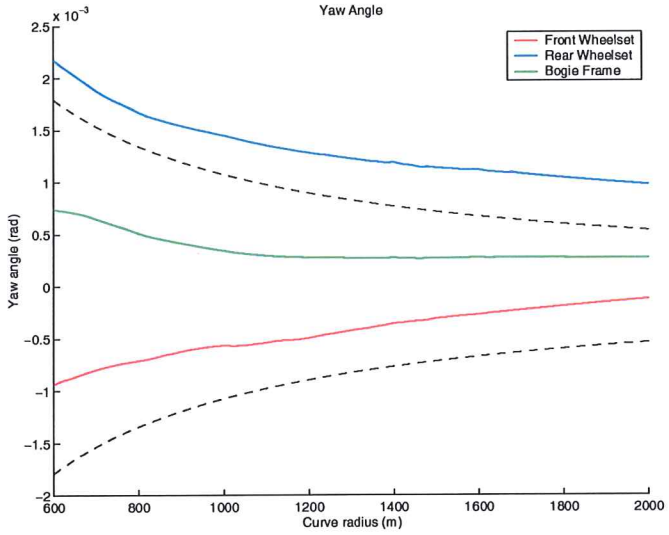


Figure 5.18: Yaw angle of the bogie as a function of the curve radius. The speed is 30 $\frac{m}{s}$ and the superelevation is 4°. The dashed lines are the ideal yaw angle of the wheelsets for which the wear is minimum.

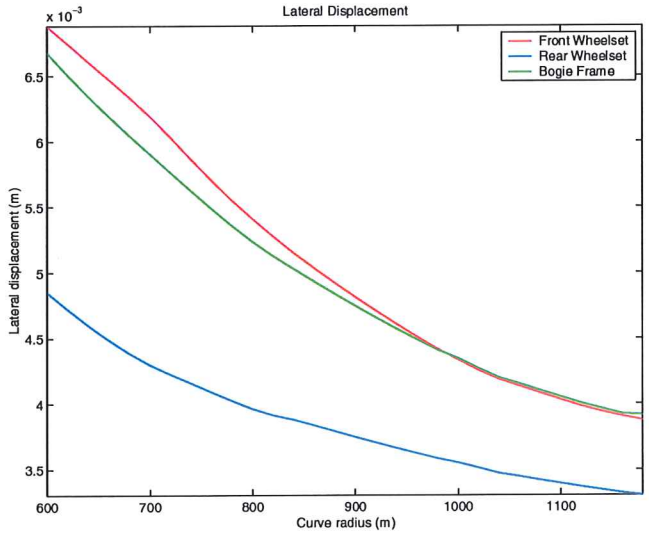


Figure 5.19: Lateral displacement of the bogie as a function of the curve radius. The speed is 50 $\frac{m}{s}$ and the superelevation is 4°.

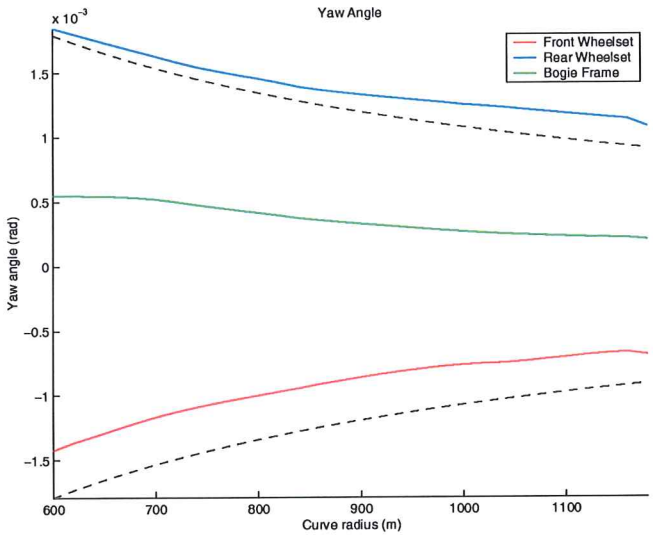


Figure 5.20: Yaw angle of the bogie as a function of the curve radius. The speed is 50 $\frac{m}{s}$ and the superelevation is 4°. The dashed lines are the ideal yaw angle of the wheelsets for which the wear is minimum.

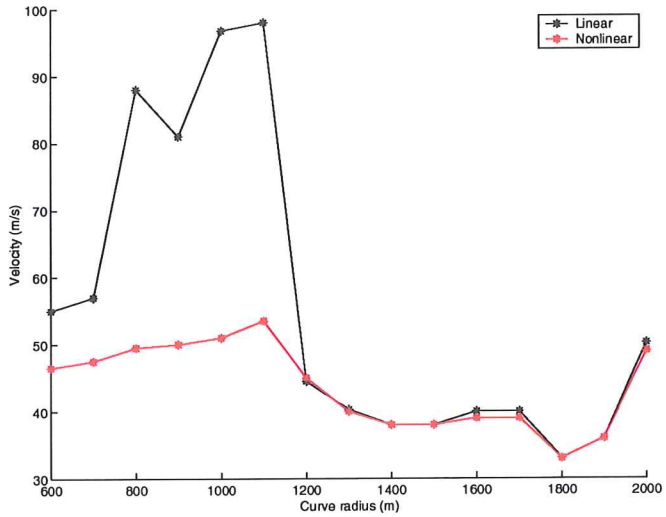


Figure 5.21: Critical velocity as a function of the curve radius. Superelevation is fixed at 2°.

decreasing, and forcing the nonlinear critical velocity to lower aswell. We also noticed that in the merged area, the hysteresis effect typical to a subcritical Hopf bifurcation no longer exists, and we have found it to be a supercritical Hopf bifurcation.

An example pointing at a supercritical Hopf bifurcation taking place is shown in figure 5.22 which shows the lateral motion of the wheelsets and bogie frame for a velocity of 36.5 m/s on a 2° superelevated track and a curve radius of 1900 m. What we see here is a time series at a velocity just slightly above the point of bifurcation. The extremely low amplitude of the hunting indicates that contrary to our solution ending up suddenly on a distant attractor, as expected for a subcritical Hopf bifurcation, the attractor grows slowly out of the previous central equilibrium position.

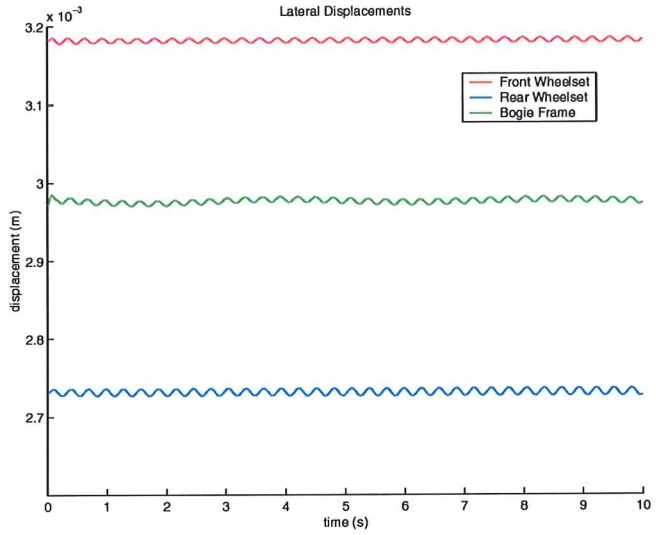


Figure 5.22: The weakly hunting bogie at $v = 36.5$ m/s, curve radius of 1900 m and a 2° superelevated track.

In locating the linear critical velocities, we proceeded as earlier described, by detecting a conjugate eigenvalue pair crossing the imaginary axis. This is shown for 2° superelevated track, and curve radius 1000 m and 1300 m, respectively, in figures 5.23 and 5.24.

In detecting the nonlinear critical velocities, we also proceeded in the same fashion as earlier. We commenced on the hunting attractor, and reduced velocity adiabatically until hunting ceased when the saddle node bifurcation was reached. For a curve radius of 600 m, and 2° superelevation, we obtained figure 5.25.

This strategy was applicable up to the point where the critical velocities

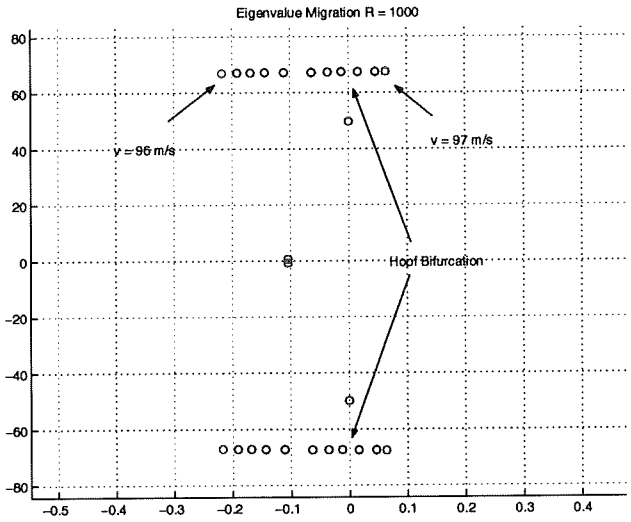


Figure 5.23: Eigenvalue migration for a curve radius of 1000 m and a 2° superelevated track.

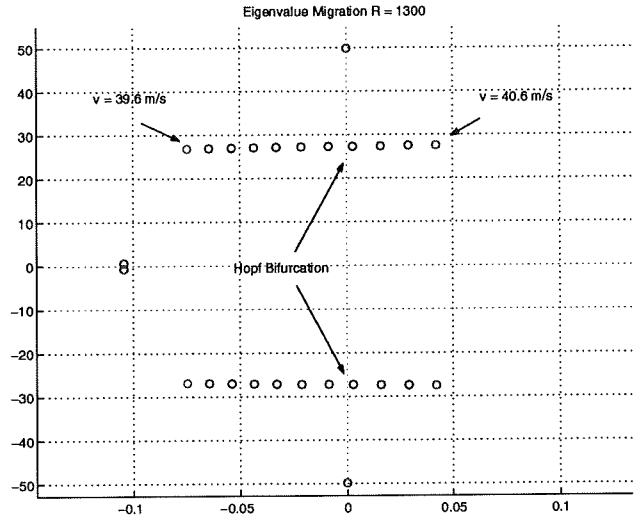


Figure 5.24: Eigenvalue migration for a curve radius of 1300 m and a 2° superelevated track.

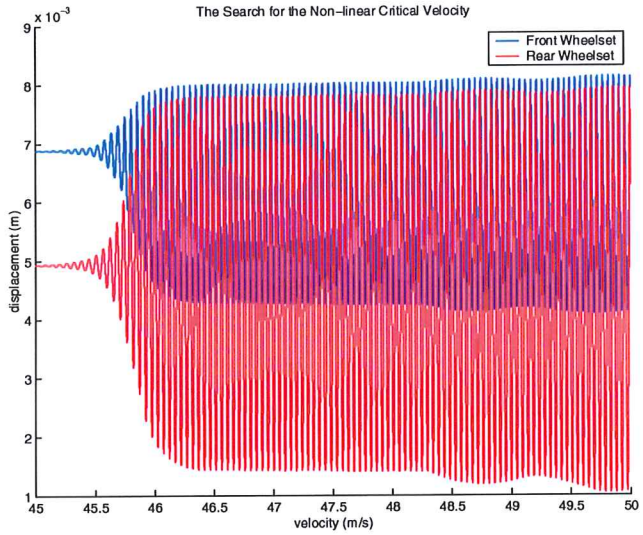


Figure 5.25: Adiabatic reduction of velocity in order to detect nonlinear critical velocity for curve radius 600 m and 2° superelevation.

merged, i.e. $R < 1200$ m. For $R > 1200$ m we have some trouble with this strategy since the hunting attractor seems to shrink with lowering velocities, and not dissappear suddenly in a saddle node bifurcation.

An example of this behaviour is given in figure 5.26. Here we have uniformly decelerated our bogie from 100 m/s down to 50 m/s on a 2000 m curve radius, 2° superelevated track. Important to note is that the attractor is truly shrinking in size, and has not vanished. Figure 5.27 shows that at 50 m/s, the bogie will still hunt if kept at that speed.

The possibility does exist that we still have a subcritical Hopf and saddle node bifurcations, but that the 'fold' is now too small to notice in our analysis.

The hysteresis effect associated with a subcritical Hopf and saddle node bifurcation is proven to exist at a curve radius of 2000 m and 2° superelevation. We showed this by entering on the hunting attractor at 50 m/s, and lowered the velocity down to 48 m/s. Figures 5.27, 5.28 and 5.29 show these steps. We see that we have now passed the saddle node bifurcation, since at 48 m/s the bogie no longer hunts. From here, we move up again to 50 m/s and still experience no hunting motion as seen in figure 5.30.

5.2.3 Other Results

An important factor to consider in our experiments, is the influence of the car body rolling on top of the bogie. We experienced this in various simulations,

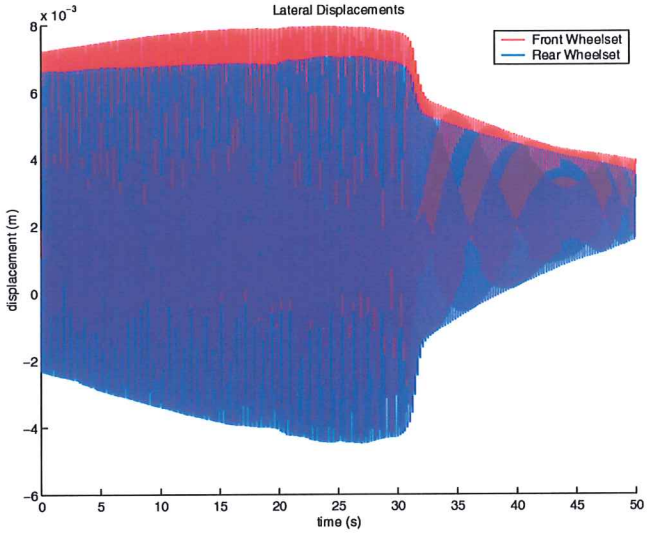


Figure 5.26: Lateral motions of front and rear wheelsets when uniformly decelerated from 100 m/s to 50 m/s over 50 seconds. Important to note is that the attractor is truly shrinking in size, and has not vanished.

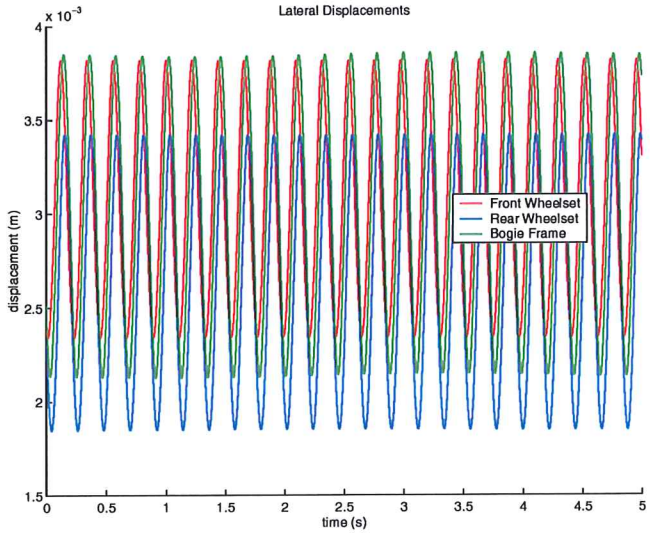


Figure 5.27: Lateral motion of the wheelsets and bogie frame at 50 m/s, 2000 m curve radius and 2° superelevated track. The bogie is on the hunting attractor.

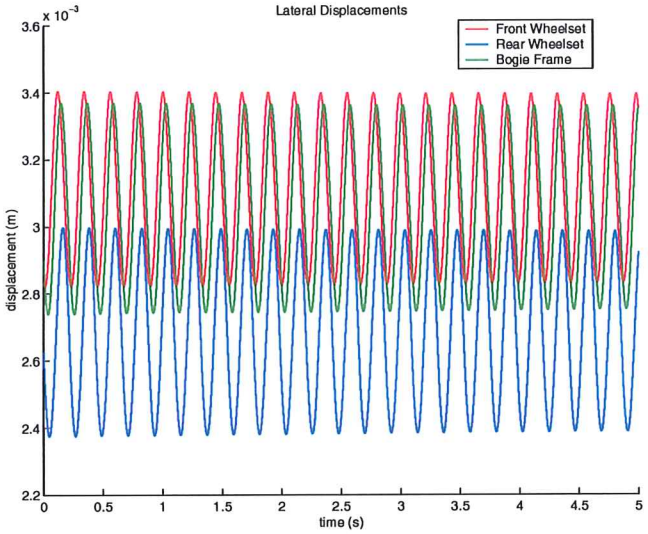


Figure 5.28: Lateral motion of the wheelsets and bogie frame at 49 m/s, 2000 m curve radius and 2° superelevated track. The bogie is still on the hunting attractor.

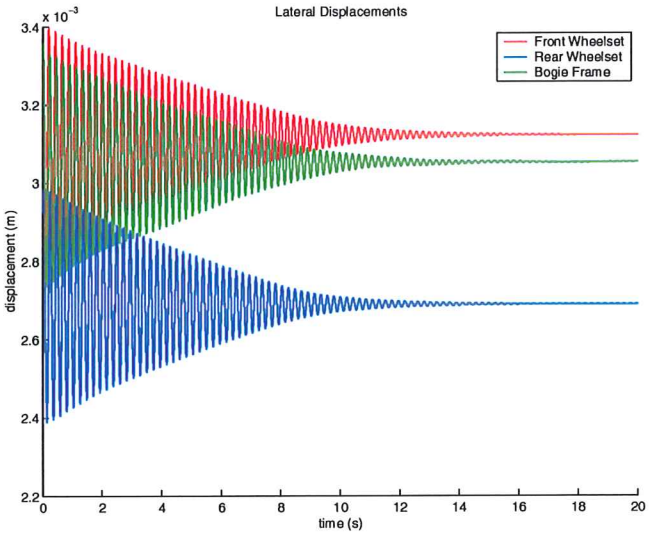


Figure 5.29: Lateral motion of the wheelsets and bogie frame at 48 m/s, 2000 m curve radius and 2° superelevated track. The hunting attractor no longer exists, and the bogie is attracted to the non-oscillating equilibrium point.

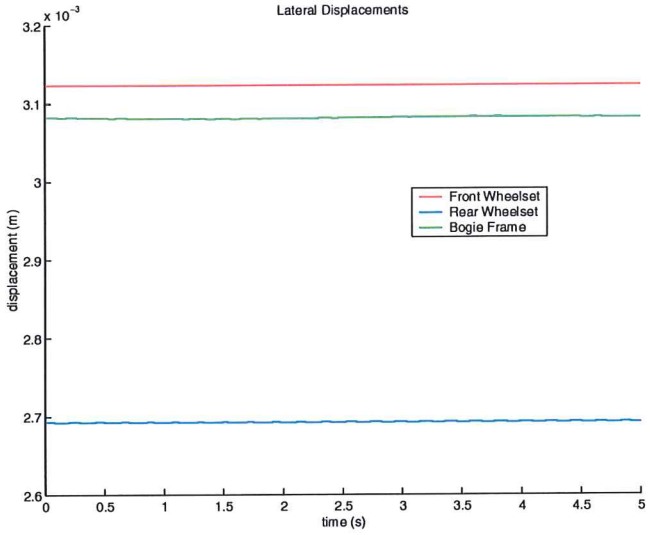


Figure 5.30: Lateral motion of the wheelsets and bogie frame at 50 m/s, 2000 m curve radius and 2° superelevated track. The bogie remains on the non-oscillating equilibrium point.

and the persistence of the phenomenon was attributable to the strength of the dampers connecting the car body to the bogie frame (D_1). In order to remove this oscillation, we strengthened the dampers, ran the simulation for some time, and then reset the dampers to their original strengths. Figures 5.31 and 5.32 illustrate the effect of car roll. This effect is most noticeable on the bogie frame, and not so much on the wheelsets, as expected. Easily noticed, is the fact that the low frequency motion of the bogie frame matches the frequency of the car roll. We also notice that the roll of the car body is negative throughout the simulation, meaning that the centrifugal force is forcing the car body outwards in the curve.

Another interesting result is to see how the wheelsets align themselves on the track for varying parameters. We here compare the alignments for different velocities on a 2° superelevated track with a curve radius of 1000 m. Previous figures have shown the alignment values for $v = 30$ and 50 m/s. Figures 5.33, 5.34, 5.35, and 5.36 show these values for $v = 70$ and 90 m/s. Ultimately, figure 5.37 schematically shows the alignments.

The tendency seems correct in that as the bogie is forced outward in the curve, the wheels will tend to turn to counteract the outward tendency.

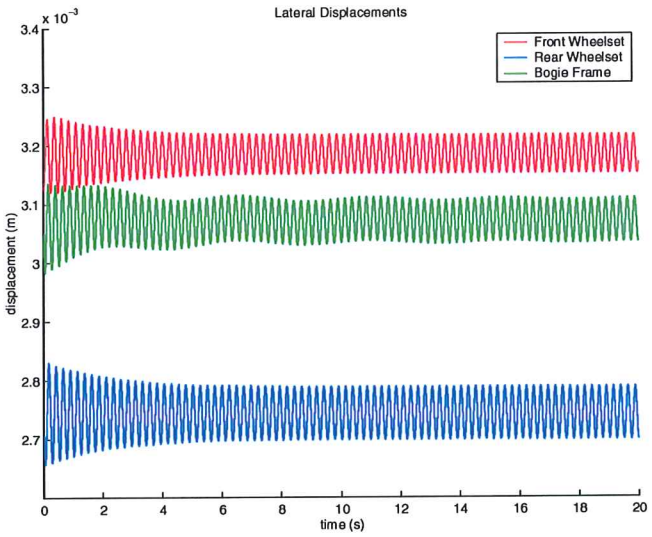


Figure 5.31: Lateral motions of the bogie elements under the influence of the rolling car body. The parameters are $v = 44$ m/s, 1900 m curve radius and 2° superelevation.

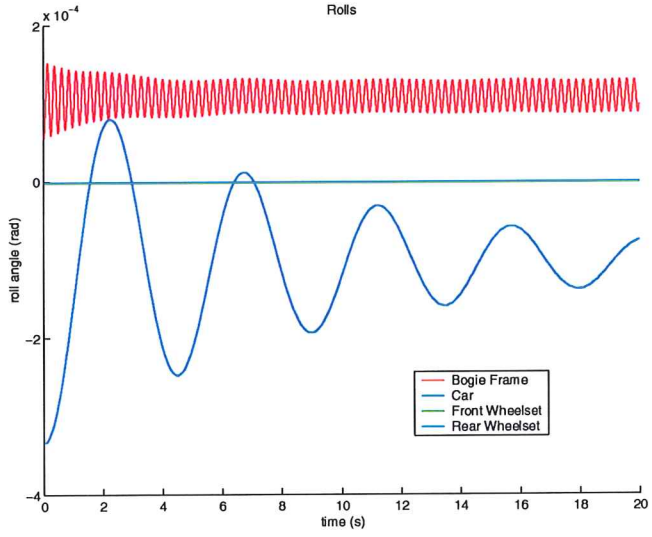


Figure 5.32: Rolling motions of the bogie elements, as well as the car body roll. The parameters are $v = 44$ m/s, 1900 m curve radius and 2° superelevation.

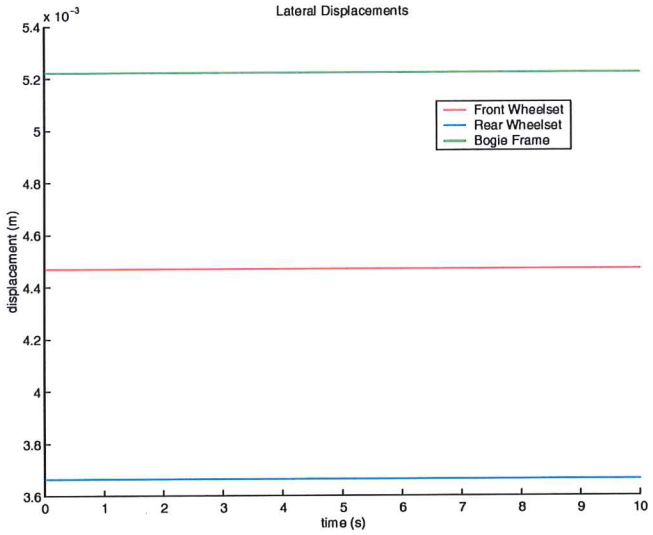


Figure 5.33: Lateral bogie element movements for $v = 70$ m/s. Track curve radius set at 1000 m and with a 2° superelevation.

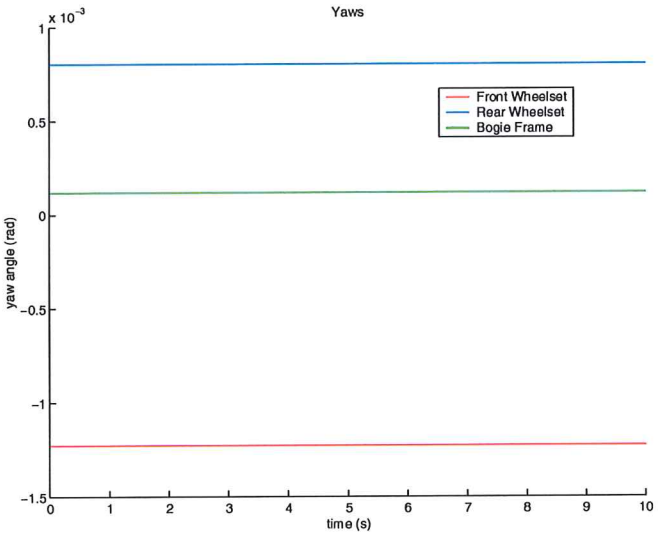


Figure 5.34: Bogie element yaw movements for $v = 70$ m/s. Track curve radius set at 1000 m and with a 2° superelevation.

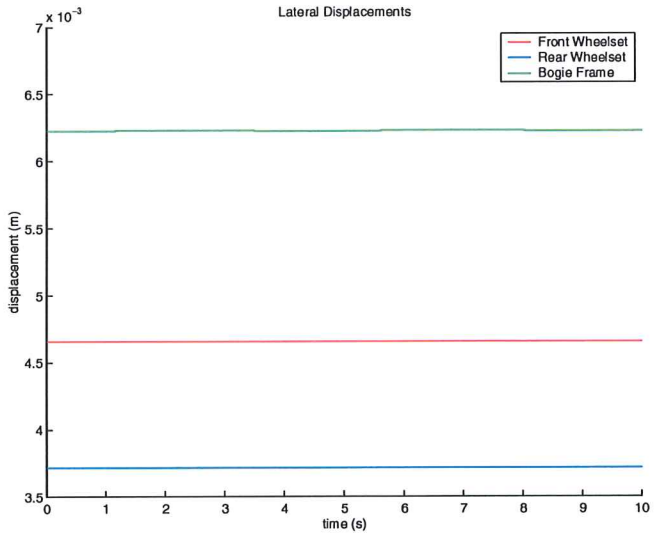


Figure 5.35: Lateral bogie element movements for $v = 90$ m/s. Track curve radius set at 1000 m and with a 2° superelevation.

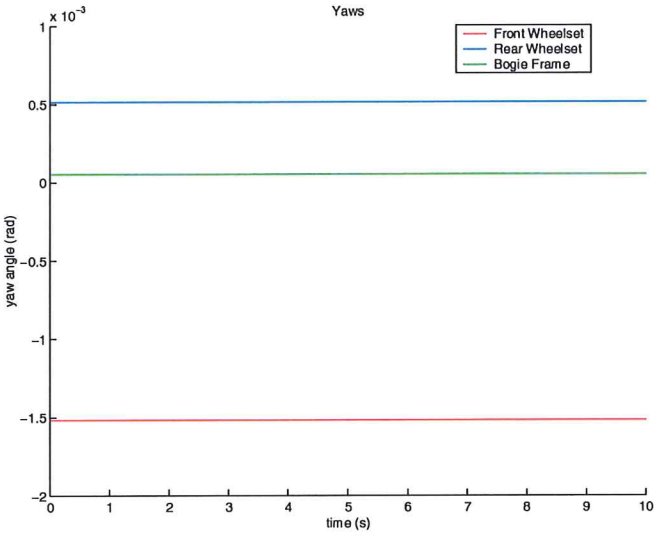


Figure 5.36: Bogie element yaw movements for $v = 90$ m/s. Track curve radius set at 1000 m and with a 2° superelevation.

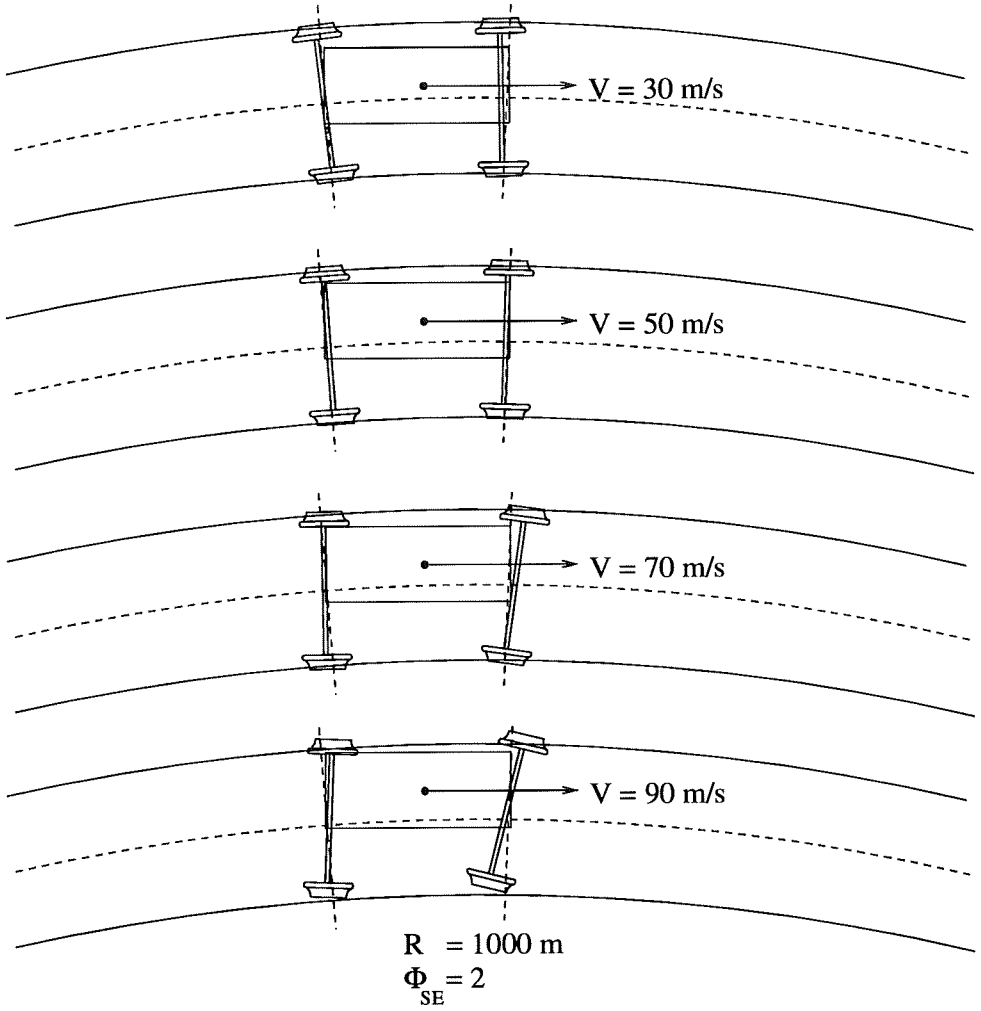


Figure 5.37: Illustration of the wheelset and bogie frame alignments for v = 30, 50, 70 and 90 m/s. Track curve radius set at 1000 m and with a 2° superelevation.

Chapter 6

Remarks

Clearly, the greatest compromises taken under the development of our model, was in relation to the wheel/rail contact physics. This was done out of necessity (lack of computer power), and the fact that certain approximations still yield physical phenomena in our model which do approach realistic values, more or less.

One of the compromises was modelling wheel/rail deformation with a penetration model. The problem is that we wish to take the vertical degrees of freedom into account, by updating the static penetration provided by the RSGEO program. This is a crucial step in our model since approximation errors has a big influence on the resulting normal force, due to the fact that the penetration is quite small, making the additional penetration from the vertical degrees of freedom even smaller. The problem with this is at too great timesteps, the jumps in normal forces become too great for the model to handle.

Ideally, the contact forces should be found dynamically by analysing the stresses in the contact patch, and then use a finite-differences or finite-element method to find the forces given any variation of the degrees of freedom. The disadvantage with this is that we go from a finite dimensional system to an infinite-dimensional system, and it will yield a prohibitive cost on computation time in order to solve.

Chapter 7

Conclusion

In this project, we have managed to establish a mathematical model for the dynamics of a railcar bogie based on Cooperrider's bogie. We have equipped the model with the ability of running the bogie on a superelevated curved track.

The contact physics between the wheel and the rail were modelled through Hertz' theory. Wheel/rail deformation are modelled as penetration of the wheel into the rail, and associated tangential forces are calculated through the method presented by Shen, Hedrick and Elkins.

The implemented program has been documented and explained, and it should not be an impossible task to pick it up and go from where we left off. A good deal of time and effort were applied in making an easily understandeable program structure.

With regards to results, we managed to show how the critical velocities were reduced in curved tracks, but were surprised to encounter a sudden period of robustness as curve radius decreased. This may be attributable to centrifugal forces stabilizing the system.

We have also seen how the whole bogie construction tends to settle into a certain configuration in a curve, with respect to lateral displacement and yaw angles.

Appendix A

Definitions

- a_e
  Major semi axis in the contact ellipse.
- a_{fl}
  Distance between the center of mass of the front wheelset and the contact point of the front left wheel.
- a_{fr}
  Distance between the center of mass of the front wheelset and the contact point of the front right wheel.
- a_{rl}
  Distance between the center of mass of the rear wheelset and the contact point of the rear left wheel.
- a_{rr}
  Distance between the center of mass of the rear wheelset and the contact point of the rear right wheel.
- a_{Rl}
  Lateral distance between the contact point and the center of the track for the left wheel.
- a_{Rr}
  Lateral distance between the contact point and the center of the track for the right wheel.
- b_e
  Minor semi axis in the contact ellipse.
- δ_l
  Conicity for the left wheel.
- δ_r
  Conicity for the right wheel.
- Δq_l
  Additional penetration on the left wheel.

- Δq_r
  Additional penetration on the right wheel.
- h_1
  Distance from the primary suspension to the center of mass of the bogie frame.
- h_2
  Distance from the center of mass of the bogie frame to the secondary suspension.
- h_3
  Distance from the secondary suspension to the rotation point of the car body.
- h_4
  Distance from the rotation point of the car body to the the center of mass of the car body.
- I_{cx}
  Moment of inertia of car body around x.
- I_{fx}
  Moment of inertia of bogie frame around x.
- I_{fy}
  Moment of inertia of bogie frame around y.
- I_{fz}
  Moment of inertia of bogie frame around z.
- I_{wx}
  Moment of inertia of wheelset around x.
- I_{wy}
  Moment of inertia of wheelset around y.
- I_{wz}
  Moment of inertia of wheelset around z.
- m_c
  Mass of car body.
- m_f
  Mass of bogie frame.
- m_w
  Mass of wheelset.
- μ
  Coefficient of friction between wheel and rail. Value set to 0.15 for our experiments.
- Ω
  Angular velocity of the wheelset around y. It is related to the angular velocity perturbation found in the differential equations by $\Omega = \frac{V}{r_0} + \beta$.
- Φ
  Roll angle. Rotation around x.
- Φ_{se}
  Superelevation.

- Ψ
  Yaw angle. Rotation around z.
- q_0
  Static penetration as delivered by RSGEO.
- r_{fl}
  Rolling radius for the front left wheel.
- r_{fr}
  Rolling radius for the front right wheel.
- r_{rl}
  Rolling radius for the rear left wheel.
- r_{rr}
  Rolling radius for the rear right wheel.
- R
  Curve radius.
- r_0
  Basic rolling radius.
- Θ
  Pitch angle. Rotation around y.
- V
  Velocity of the bogie.
- ξ
  Creep.
- z_{Rl}
  The vertical distance of the contact point of the left wheel with respect to the rail coordinate system.
- z_{Rr}
  The vertical distance of the contact point of the right wheel with respect to the rail coordinate system.

Appendix B

Coordinate Transformations

In this chapter we will derive the transformation matrices which provide the connection between the coordinate systems we have used to describe the model. This is a matter of changing basis in a three dimensional Euclidean vector space.

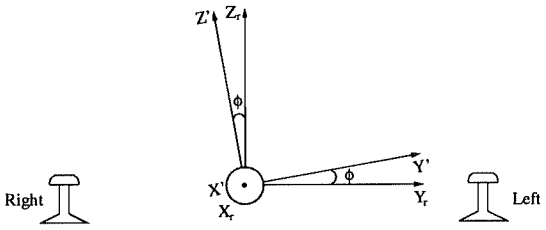


Figure B.1: Rotate the angle ϕ around x

First we will find the transformation matrix between the rail coordinate system and the wheelset coordinate system. We rotate the rail coordinate system with the angle ϕ around its own x axis, and end up with the coordinate system denoted (X', Y', Z') (see figure B.1). Let the canonical basis for the rail and (X', Y', Z') coordinate system be $\{\mathbf{e}_{xr}, \mathbf{e}_{yr}, \mathbf{e}_{zr}\}$ and $\{\mathbf{e}_{x'}, \mathbf{e}_{y'}, \mathbf{e}_{z'}\}$, respectively. The matrix M_ϕ, that switches rail coordinates to (X', Y', Z') coordinates is the coordinate matrix for $\{\mathbf{e}_{xr}, \mathbf{e}_{yr}, \mathbf{e}_{zr}\}$ with respect to the basis $\{\mathbf{e}_{x'}, \mathbf{e}_{y'}, \mathbf{e}_{z'}\}$, which is

$$\mathbf{e}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{e}_y = \begin{bmatrix} 0 \\ \cos\phi \\ -\sin\phi \end{bmatrix} \quad \mathbf{e}_z = \begin{bmatrix} 0 \\ \sin\phi \\ \cos\phi \end{bmatrix}$$

i.e.

$$\mathbf{M}_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

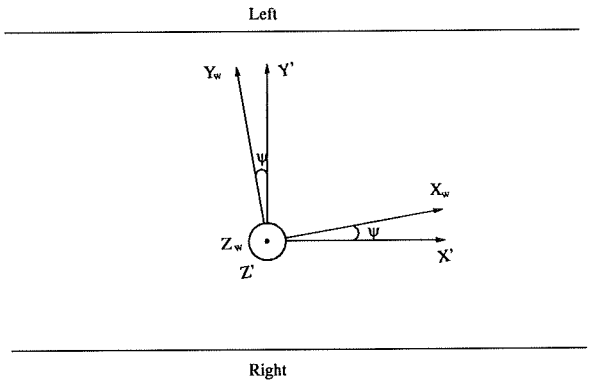


Figure B.2: Rotation ψ around z'

We now proceed by rotating the (X', Y', Z') coordinate system ψ around its own z' axis and we end up with the wheelset coordinate system. The matrix that switches (X', Y', Z') coordinates to (X_w, Y_w, Z_w) coordinates is the coordinate matrix for $\{\mathbf{e}_{x'}, \mathbf{e}_{y'}, \mathbf{e}_{z'}\}$ with respect to the basis $\{\mathbf{e}_{xw}, \mathbf{e}_{yw}, \mathbf{e}_{zw}\}$.

$$\mathbf{e}_x = \begin{bmatrix} \cos\psi \\ -\sin\psi \\ 0 \end{bmatrix} \quad \mathbf{e}_y = \begin{bmatrix} \sin\psi \\ \cos\psi \\ 0 \end{bmatrix} \quad \mathbf{e}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Thus

$$\mathbf{M}_\psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The matrix that switches rail coordinates to wheelset coordinates is

$${}_w\mathbf{M}_r = \mathbf{M}_\psi \cdot \mathbf{M}_\phi = \begin{bmatrix} \cos\psi & \sin\psi\cos\phi & \sin\psi\sin\phi \\ -\sin\psi & \cos\psi\cos\phi & \cos\psi\sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \approx \begin{bmatrix} 1 & \psi & 0 \\ -\psi & 1 & \phi \\ 0 & -\phi & 1 \end{bmatrix}$$

We now seek the transformation matrix that converts contact coordinates to wheelset coordinates and it is found in precisely the same manner as before (see figure B.3). For the left wheel the coordinate matrix for $\{\mathbf{e}_{xc}, \mathbf{e}_{yc}, \mathbf{e}_{zc}\}$

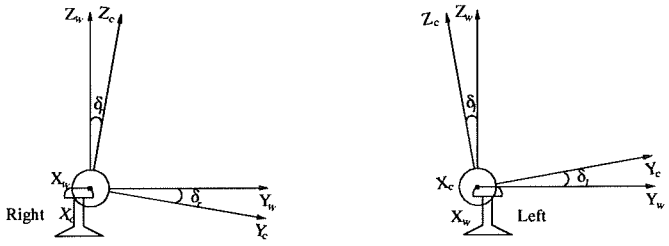


Figure B.3:

with respect to the basis $\{\mathbf{e}_{xw}, \mathbf{e}_{yw}, \mathbf{e}_{zw}\}$ is

$$\mathbf{e}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{e}_y = \begin{bmatrix} 0 \\ \cos\delta_l \\ \sin\delta_l \end{bmatrix} \quad \mathbf{e}_z = \begin{bmatrix} 0 \\ -\sin\delta_l \\ \cos\delta_l \end{bmatrix}$$

Hence

$${}_w\mathbf{M}_{cl} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_l & -\sin\delta_l \\ 0 & \sin\delta_l & \cos\delta_l \end{bmatrix}$$

For the right wheel we get

$$\mathbf{e}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{e}_y = \begin{bmatrix} 0 \\ \cos\delta_r \\ -\sin\delta_r \end{bmatrix} \quad \mathbf{e}_z = \begin{bmatrix} 0 \\ \sin\delta_r \\ \cos\delta_r \end{bmatrix}$$

Therefore

$${}_w\mathbf{M}_{cr} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_r & \sin\delta_r \\ 0 & -\sin\delta_r & \cos\delta_r \end{bmatrix}$$

An important property with the transformation matrices ${}_w\mathbf{M}_r$, ${}_w\mathbf{M}_{cl}$ and ${}_w\mathbf{M}_{cr}$ is that they are orthogonal. This means that

$$\begin{aligned} {}_r\mathbf{M}_w &= {}_w\mathbf{M}_r^{-1} = {}_w\mathbf{M}_r^T \\ {}_{cl}\mathbf{M}_w &= {}_w\mathbf{M}_{cl}^{-1} = {}_w\mathbf{M}_{cl}^T \\ {}_{cr}\mathbf{M}_w &= {}_w\mathbf{M}_{cr}^{-1} = {}_w\mathbf{M}_{cr}^T \end{aligned}$$

We are also able to find the transformation matrices that converts contact coordinates to rail coordinates and vice versa. We utilize the addition formulae and get

$${}_r\mathbf{M}_{cl} = {}_r\mathbf{M}_w \cdot {}_w\mathbf{M}_{cl} \approx \begin{bmatrix} 1 & -\psi\cos\delta_l & \psi\sin\delta_l \\ \psi & \cos(\delta_l+\phi) & -\sin(\delta_l+\phi) \\ 0 & \sin(\delta_l+\phi) & \cos(\delta_l+\phi) \end{bmatrix}$$

$$
{}_r\mathbf{M}_{cr} = {}_r\mathbf{M}_w \cdot {}_w\mathbf{M}_{cr} \approx \begin{bmatrix} 1 & -\psi\cos\delta_r & -\psi\sin\delta_r \\ \psi & \cos(\delta_r - \phi) & \sin(\delta_r - \phi) \\ 0 & -\sin(\delta_r - \phi) & \cos(\delta_r - \phi) \end{bmatrix}
$$

$$
{}_{cl}\mathbf{M}_r = {}_{cl}\mathbf{M}_w \cdot {}_w\mathbf{M}_r \approx \begin{bmatrix} 1 & \psi & 0 \\ -\psi\cos\delta_l & \cos(\delta_l + \phi) & \sin(\delta_l + \phi) \\ \psi\sin\delta_l & -\sin(\delta_l + \phi) & \cos(\delta_l + \phi) \end{bmatrix}
$$

$$
{}_{cr}\mathbf{M}_r = {}_{cr}\mathbf{M}_w \cdot {}_w\mathbf{M}_r \approx \begin{bmatrix} 1 & \psi & 0 \\ -\psi\cos\delta_r & \cos(\delta_r - \phi) & -\sin(\delta_r - \phi) \\ -\psi\sin\delta_r & \sin(\delta_r - \phi) & \cos(\delta_r - \phi) \end{bmatrix}
$$

Appendix C

Creepage

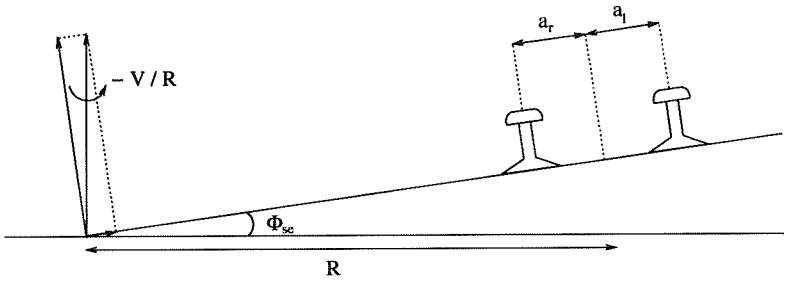


Figure C.1: The curve parameters

We consider three coordinate systems that are similar to the one defined previously, but they differ in one particular place. They are not moving with the velocity of the bogie. We denote these coordinate systems by (X'_r, Y'_r, Z'_r), (X'_w, Y'_w, Z'_w) and (X'_c, Y'_c, Z'_c) respectively, and for each contact point on the wheels we wish to find the velocity with respect to (X'_c, Y'_c, Z'_c), i.e. the velocity of the contact point relative to the rail. We note that the rotation matrices between these coordinate systems are the same as between the original coordinate system.

The velocity of the contact points consists of two parts. Firstly, there is a contribution due to the translational velocity of the wheelsets, and secondly, the angular velocity of the wheelsets contributes. This is written as

$$\mathbf{V}_c = \mathbf{V}_{w,trans} + \mathbf{V}_{w,rot}$$

The translational velocity of the center of mass of a wheelset is

$$\mathbf{V}_{w,trans} = V \cdot \mathbf{x}'_r + \dot{y} \cdot \mathbf{y}'_r + \dot{z} \cdot \mathbf{z}'_r$$

The rotational velocity of the wheelset is

$$\mathbf{V}_{w,rot} = \omega_w \times \mathbf{r}_c + \omega_{w,frame} \times \mathbf{r}_w$$

where

$$\omega_w = \dot{\phi} \cdot \mathbf{x}'_r + \Omega \cdot \mathbf{y}'_w + \dot{\psi} \cdot \mathbf{z}'_w$$

The vector from the center of mass of the wheelset to the contact point is

$$\mathbf{r}_c = 0 \cdot \mathbf{x}'_w + a^* \cdot \mathbf{y}'_w - r \cdot \mathbf{z}'_w$$

where $(a^*, r) = (a_l, r_l)$ for left contact point and $(a^*, r) = (-a_r, r_r)$ for right contact point. Furthermore, we have

$$\omega_{w,frame} = 0 \cdot \mathbf{x}'_r - \frac{V}{R} \sin \Phi_{se} \cdot \mathbf{y}'_r - \frac{V}{R} \cos \Phi_{se} \cdot \mathbf{z}'_r$$

and

$$\mathbf{r}_w = 0 \cdot \mathbf{x}'_r + a^{**} \cdot \mathbf{y}'_r + 0 \cdot \mathbf{z}'_r$$

where $a^{**} = a$ for left contact point and $a^{**} = -a$ for right contact point (see figure C.1 for a). This $\omega_{w,frame}$ is due to the fact that our original rail frame (X_r, Y_r, Z_r) is continuously rotating with this $\omega_{w,frame}$ in order to follow the curve. The terms provides the difference in velocity one would expect between the left and right contact points, since they are rotating with the same angular velocity through the curve, but with different radius (see figure C.1). In reference to (X'_w, Y'_w, Z'_w) we have

$$\begin{aligned}
\mathbf{V}_c &= {}_w\mathbf{M}_r \cdot \begin{bmatrix} V \\ \dot{y} \\ \dot{z} \end{bmatrix} + \left({}_w\mathbf{M}_r \cdot \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \Omega \\ \dot{\psi} \end{bmatrix} \right) \times \begin{bmatrix} 0 \\ a^* \\ -r \end{bmatrix} \\
&\quad + {}_w\mathbf{M}_r \cdot \left(\begin{bmatrix} 0 \\ -\frac{V}{R} \sin \Phi_{se} \\ -\frac{V}{R} \cos \Phi_{se} \end{bmatrix} \times \begin{bmatrix} 0 \\ a^{**} \\ 0 \end{bmatrix} \right) \\
&= \begin{bmatrix} V + \psi\dot{y} \\ -\psi V + \phi\dot{z} + \dot{y} \\ -\phi\dot{y} + \dot{z} \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ \Omega - \psi\dot{\phi} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} 0 \\ a^* \\ -r \end{bmatrix} + \begin{bmatrix} \frac{a^{**}V}{R} \cos \Phi_{se} \\ -\frac{\psi a^{**}V}{R} \cos \Phi_{se} \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} V + \psi\dot{y} + r(\psi\dot{\phi} - \Omega) - a^*\dot{\psi} + \frac{a^{**}V}{R} \cos \Phi_{se} \\ -\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r - \frac{\psi a^{**}V}{R} \cos \Phi_{se} \\ -\phi\dot{y} + \dot{z} + a^*\dot{\phi} \end{bmatrix}
\end{aligned}$$

In reference to (X_c', Y_c', Z_c') we now have

$$\begin{aligned}
\mathbf{V}_{cl} &= {}_{cl}\mathbf{M}_w \cdot \begin{bmatrix} V + \psi\dot{y} + r_l(\psi\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se} \\ -\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_l - \frac{\psi aV}{R}\cos\Phi_{se} \\ -\phi\dot{y} + \dot{z} + a_l\dot{\phi} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_l & \sin\delta_l \\ 0 & -\sin\delta_l & \cos\delta_l \end{bmatrix} \cdot \begin{bmatrix} V + \psi\dot{y} + r_l(\psi\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se} \\ -\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_l - \frac{\psi aV}{R}\cos\Phi_{se} \\ -\phi\dot{y} + \dot{z} + a_l\dot{\phi} \end{bmatrix} \\
&= \begin{bmatrix} V + \psi\dot{y} + r_l(\psi\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se} \\ \left(-\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_l - \frac{\psi aV}{R}\cos\Phi_{se}\right)\cos\delta_l + \left(-\phi\dot{y} + \dot{z} + a_l\dot{\phi}\right)\sin\delta_l \\ -\left(-\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_l - \frac{\psi aV}{R}\cos\Phi_{se}\right)\sin\delta_l + \left(-\phi\dot{y} + \dot{z} + a_l\dot{\phi}\right)\cos\delta_l \end{bmatrix} \\
&= \begin{bmatrix} V + \psi\dot{y} + r_l(\psi\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se} \\ \left(-\psi V\left(1 + \frac{a}{R}\cos\Phi_{se}\right) + \phi\dot{z} + \dot{y} + \dot{\phi}r_l\right)\cos\delta_l + \left(-\phi\dot{y} + \dot{z} + a_l\dot{\phi}\right)\sin\delta_l \\ -\left(-\psi V\left(1 + \frac{a}{R}\cos\Phi_{se}\right) + \phi\dot{z} + \dot{y} + \dot{\phi}r_l\right)\sin\delta_l + \left(-\phi\dot{y} + \dot{z} + a_l\dot{\phi}\right)\cos\delta_l \end{bmatrix} \\
\mathbf{V}_{cr} &= {}_{cr}\mathbf{M}_w \cdot \begin{bmatrix} V + \psi\dot{y} + r_r(\psi\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se} \\ -\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_r + \frac{\psi aV}{R}\cos\Phi_{se} \\ -\phi\dot{y} + \dot{z} - a_r\dot{\phi} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_r & -\sin\delta_r \\ 0 & \sin\delta_r & \cos\delta_r \end{bmatrix} \cdot \begin{bmatrix} V + \psi\dot{y} + r_r(\psi\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se} \\ -\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_r + \frac{\psi aV}{R}\cos\Phi_{se} \\ -\phi\dot{y} + \dot{z} - a_r\dot{\phi} \end{bmatrix} \\
&= \begin{bmatrix} V + \psi\dot{y} + r_r(\psi\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se} \\ \left(-\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_r + \frac{\psi aV}{R}\cos\Phi_{se}\right)\cos\delta_r - \left(-\phi\dot{y} + \dot{z} - a_r\dot{\phi}\right)\sin\delta_r \\ \left(-\psi V + \phi\dot{z} + \dot{y} + \dot{\phi}r_r + \frac{\psi aV}{R}\cos\Phi_{se}\right)\sin\delta_r + \left(-\phi\dot{y} + \dot{z} - a_r\dot{\phi}\right)\cos\delta_r \end{bmatrix} \\
&= \begin{bmatrix} V + \psi\dot{y} + r_r(\psi\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se} \\ \left(-\psi V\left(1 - \frac{a}{R}\cos\Phi_{se}\right) + \phi\dot{z} + \dot{y} + \dot{\phi}r_r\right)\cos\delta_r - \left(-\phi\dot{y} + \dot{z} - a_r\dot{\phi}\right)\sin\delta_r \\ \left(-\psi V\left(1 - \frac{a}{R}\cos\Phi_{se}\right) + \phi\dot{z} + \dot{y} + \dot{\phi}r_r\right)\sin\delta_r + \left(-\phi\dot{y} + \dot{z} - a_r\dot{\phi}\right)\cos\delta_r \end{bmatrix}
\end{aligned}$$

Figure (C.2) shows that the wheelset axles are not perpendicular to the rails as they would be for a straight track. We take this effect into account by adjusting the yaw angle.

$$\psi \rightarrow \psi + \alpha$$

where $\alpha = \frac{b}{R}$ for the front wheelset and $\alpha = -\frac{b}{R}$ for the rear wheelset, and we note that $\dot{\psi}$ does not change under this transformation, since α is a constant. We are now able to write up the longitudinal and lateral creep terms. For

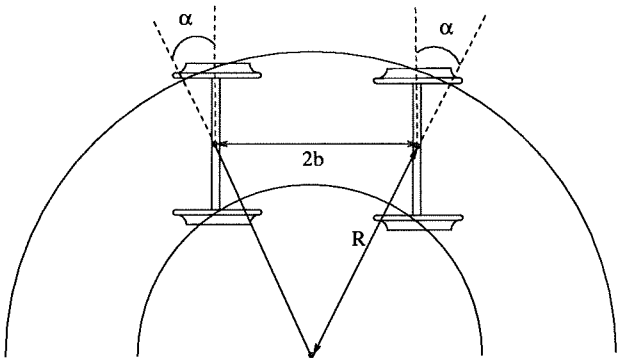


Figure C.2: Wheelsets in centered position when the bogie is going through a curve

the front wheelset we get

$$\xi_{flx} = \frac{V + (\psi + \frac{b}{R})\dot{y} + r_l((\psi + \frac{b}{R})\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se}}{V}$$

$$\xi_{frx} = \frac{V + (\psi + \frac{b}{R})\dot{y} + r_r((\psi + \frac{b}{R})\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se}}{V}$$

$$\xi_{fly} = \frac{(-V(\psi + \frac{b}{R})(1 + \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_l)\cos\delta_l + (-\phi\dot{y} + \dot{z} + a_l\dot{\phi})\sin\delta_l}{V}$$

$$\xi_{fry} = \frac{(-V(\psi + \frac{b}{R})(1 - \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_r)\cos\delta_r - (-\phi\dot{y} + \dot{z} - a_r\dot{\phi})\sin\delta_r}{V}$$

and for the rear wheelset we use

$$\xi_{rlx} = \frac{V + (\psi - \frac{b}{R})\dot{y} + r_l((\psi - \frac{b}{R})\dot{\phi} - \Omega) - a_l\dot{\psi} + \frac{aV}{R}\cos\Phi_{se}}{V}$$

$$\xi_{rrx} = \frac{V + (\psi - \frac{b}{R})\dot{y} + r_r((\psi - \frac{b}{R})\dot{\phi} - \Omega) + a_r\dot{\psi} - \frac{aV}{R}\cos\Phi_{se}}{V}$$

$$\xi_{rly} = \frac{(-V(\psi - \frac{b}{R})(1 + \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_l)\cos\delta_l + (-\phi\dot{y} + \dot{z} + a_l\dot{\phi})\sin\delta_l}{V}$$

$$\xi_{rry} = \frac{(-V(\psi - \frac{b}{R})(1 - \frac{a}{R}\cos\Phi_{se}) + \phi\dot{z} + \dot{y} + \dot{\phi}r_r)\cos\delta_r - (-\phi\dot{y} + \dot{z} - a_r\dot{\phi})\sin\delta_r}{V}$$

Finally, the spin creep terms are defined as the rotation around the normal to the contact plane normalized by the velocity. First we need to find the

angular velocity of the wheelset in reference to (X'_c, Y'_c, Z'_c).

$$\begin{aligned}\omega_{c,left} &= {}_{cl}\mathbf{M}_w \cdot \omega_w \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_l & \sin\delta_l \\ 0 & -\sin\delta_l & \cos\delta_l \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \Omega - \psi\dot{\phi} \\ \dot{\psi} \end{bmatrix} \\ &= \begin{bmatrix} \dot{\phi} \\ (\Omega - \psi\dot{\phi})\cos\delta_l + \dot{\psi}\sin\delta_l \\ -(\Omega - \psi\dot{\phi})\sin\delta_l + \dot{\psi}\cos\delta_l \end{bmatrix}\end{aligned}$$

$$\begin{aligned}\omega_{c,right} &= {}_{cr}\mathbf{M}_w \cdot \omega_w \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\delta_r & -\sin\delta_r \\ 0 & \sin\delta_r & \cos\delta_r \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \Omega - \psi\dot{\phi} \\ \dot{\psi} \end{bmatrix} \\ &= \begin{bmatrix} \dot{\phi} \\ (\Omega - \psi\dot{\phi})\cos\delta_r - \dot{\psi}\sin\delta_r \\ (\Omega - \psi\dot{\phi})\sin\delta_r + \dot{\psi}\cos\delta_r \end{bmatrix}\end{aligned}$$

Thus

$$\begin{aligned}\xi_{fls} &= \frac{-(\Omega - (\psi + \frac{b}{R})\dot{\phi})\sin\delta_l + \dot{\psi}\cos\delta_l}{V} \\ \xi_{frs} &= \frac{(\Omega - (\psi + \frac{b}{R})\dot{\phi})\sin\delta_r + \dot{\psi}\cos\delta_r}{V} \\ \xi_{rls} &= \frac{-(\Omega - (\psi - \frac{b}{R})\dot{\phi})\sin\delta_l + \dot{\psi}\cos\delta_l}{V} \\ \xi_{rrs} &= \frac{(\Omega - (\psi - \frac{b}{R})\dot{\phi})\sin\delta_r + \dot{\psi}\cos\delta_r}{V}\end{aligned}$$

Appendix D

Additional Penetration

In this chapter we will derive the expressions we have used to calculate the additional penetration due to the roll and vertical motion of the wheelset. This additional penetration combined with the static penetration from the RSGEO table enables us to compute the normal force dynamically.

Firstly, we find the vector from the contact point on the wheel to the contact point on the rail. For the left wheel we get

$$\begin{aligned}\mathbf{R}_R - \mathbf{R}_w &= \begin{bmatrix} 0 \\ a_{Rl} \\ z_{Rl} \end{bmatrix} - \left(\begin{bmatrix} 0 \\ y \\ z \end{bmatrix} + {}_r\mathbf{M}_w \cdot \begin{bmatrix} 0 \\ a_l \\ -r_l \end{bmatrix} \right) \\ &= \begin{bmatrix} \psi a_l \\ a_{Rl} - y - a_l - \phi r_l \\ z_{Rl} - z + r_l - \phi a_l \end{bmatrix}\end{aligned}$$

The term $z_{Rl} + r_l$ is provided by RSGEO, since it is a static penetration. We exclude this term, because we wish to find the additional penetration. Furthermore, we need to switch to the contact coordinate system, since the penetration is the z-component of this.

$$\begin{aligned}\Delta\mathbf{P} &= {}_{cl}\mathbf{M}_r \cdot \begin{bmatrix} \psi a_l \\ a_{Rl} - y - a_l - \phi r_l \\ -z - \phi a_l \end{bmatrix} \\ &= \begin{bmatrix} 1 & \psi & 0 \\ -\psi\cos\delta_l & \cos(\delta_l + \phi) & \sin(\delta_l + \phi) \\ \psi\sin\delta_l & -\sin(\delta_l + \phi) & \cos(\delta_l + \phi) \end{bmatrix} \cdot \begin{bmatrix} \psi a_l \\ a_{Rl} - y - a_l - \phi r_l \\ -z - \phi a_l \end{bmatrix}\end{aligned}$$

The additional penetration is

$$\begin{aligned}\Delta P_{add,l} &= \psi^2 a_l \sin\delta_l - (a_{Rl} - y - a_l - \phi r_l)\sin(\delta_l + \phi) \\ &\quad +(-z - \phi a_l)\cos(\delta_l + \phi) \\ &\approx -(a_{Rl} - y - a_l - \phi r_l)\sin(\delta_l + \phi) + (-z - \phi a_l)\cos(\delta_l + \phi)\end{aligned}$$

We follow the same procedure for the right wheel and we get

$$\begin{aligned}\mathbf{R}_R - \mathbf{R}_w &= \begin{bmatrix} 0 \\ -a_{Rr} \\ z_{Rr} \end{bmatrix} - \left(\begin{bmatrix} 0 \\ y \\ z \end{bmatrix} + {}_r\mathbf{M}_w \cdot \begin{bmatrix} 0 \\ -a_r \\ -r_r \end{bmatrix} \right) \\ &= \begin{bmatrix} -\psi a_r \\ -a_{Rr} - y + a_r - \phi r_r \\ z_{Rr} - z + r_r + \phi a_r \end{bmatrix}\end{aligned}$$

$$\begin{aligned}\Delta\mathbf{P} &= {}_{cr}\mathbf{M}_r \cdot \begin{bmatrix} -\psi a_r \\ -a_{Rr} - y + a_r - \phi r_r \\ -z + \phi a_r \end{bmatrix} \\ &= \begin{bmatrix} 1 & \psi & 0 \\ -\psi\cos\delta_r & \cos(\delta_r - \phi) & -\sin(\delta_r - \phi) \\ -\psi\sin\delta_r & \sin(\delta_r - \phi) & \cos(\delta_r - \phi) \end{bmatrix} \cdot \begin{bmatrix} -\psi a_r \\ -a_{Rr} - y + a_r - \phi r_r \\ -z + \phi a_r \end{bmatrix}\end{aligned}$$

The additional penetration is

$$\begin{aligned}\Delta P_{add,r} &= \psi^2 a_r \sin\delta_r + (-a_{Rr} - y + a_r - \phi r_r)\sin(\delta_r - \phi) \\ &\quad +(-z + \phi a_r)\cos(\delta_r - \phi) \\ &\approx (-a_{Rr} - y + a_r - \phi r_r)\sin(\delta_r - \phi) + (-z + \phi a_r)\cos(\delta_r - \phi)\end{aligned}$$

Appendix E

C++ Source Code

E.1 model.cc

```
#include <fstream.h>
#include <math.h>
#include "sdirk.h"
#include <stdio.h>

#define N_VAR 30
#define N_COLS_RSGEO 13
#define N_ROWS_RSGEO 3401

//Prototypes
void fun(double,DVector &,DVector &);
void jac(double,DVector &,DMatrix &);
double rhs(double,DVector &,int);
void find_contact_forces(double,DVector &,int);
void fast_num(double,DVector &,DMatrix &);
void jac_numerical(double,DVector &,DMatrix &,int,int [],int);
void read_rsgeo();
void find_rsgeodata(double,DVector &,int);
void dynamic_update(double,DVector &,int);
void dynamic_aux(double,double,double,double,double,double,double,double,
                 int,int);
void SHE(double,DVector &,int,int,double*);
double xi_flx(double,double,double,double,double,double,double);
double xi_frx(double,double,double,double,double,double,double);
double xi_fly(double,double,double,double,double,double,double,double);
double xi_fry(double,double,double,double,double,double,double,double);
double xi_fls(double,double,double,double,double);
double xi_frs(double,double,double,double,double);
double xi_rlx(double,double,double,double,double,double,double);
double xi_rrx(double,double,double,double,double,double,double);
double xi_rly(double,double,double,double,double,double,double,double);
```

```
double xi_rry(double,double,double,double,double,double,double,double);
double xi_rls(double,double,double,double,double);
double xi_rrs(double,double,double,double,double);
double Fx(double,double,double,double);
double Fy(double,double,double,double,double,double);
void toFile(double,DVector &);
void print_rsgeo();
void print_jac(double,DVector &,int);
void latyaw_analyse();
void output_last(DVector &);
void output_last_with_y(DVector &);
void use_last(DVector &);

//Parameters
double v;      //Bogie velocity
double phi_SE; //Track superelevation
double R;      //Track curve radius

//Constants
const double
  k1=1823000.0,
  k2=3646000.0,
  k3=3646000.0,
  k4=182300.0,
  k5=333300.0,
  k6=2710000.0,
  D1=20000.0,
  D2=29200.0,
  D6=500e3,
  d1=0.620,
  d2=0.680,
  d1d1=d1*d1,
  d2d2=d2*d2,
  a=0.75,
  b=1.074,
  h1=0.0762,
  h2=0.6584,
  h3=0.8654,
  h4 = 0.9-h3,
  mw=1022.0,
  Iwy=80.0,
  Iwx=678.0,
  Iwz=678.0,
  mf=2918.9,
  Ifz=6780.0,
  Ifx=6780.0,
  Ify=6780.0,
  mrc=44388.0,
  Icx=2.80e5,
```

```
  mx=0.25*mrc+0.5*mf+mw,
  g=9.82,
  mu=0.15,
  G=2.1e11/(2*(1+.027)),
  r0 = 0.425; //Basic rolling radius.

//Global Variables
double rsgeo_table[N_ROWS_RSGEO][N_COLS_RSGEO];
double rsgeodata[2][2][N_COLS_RSGEO];
double c_forces[2][2][3]; //with respect to the initial frame of reference
double n_forces[2][2][2]; //with respect to the initial frame of reference
double Omega_0; //Basic angular velocity of the wheel
Sdirk *MyInstance;
ofstream sol("sol.dat"); //Solution file
ofstream forces("forces.dat"); //Force file
ofstream latyaw("latyaw.dat"); //Lat Yaw file
bool recalc = true;
bool go_cout = false;

enum wheelset {FRONT,REAR};
enum wheel {LEFT,RIGHT};
enum cf {FX,FY,FZ};
enum nf {NY,NZ};
enum rsgeo_values {RSG_lat,RSG_N,RSG_angle,RSG_a,RSG_b,RSG_Kwy,
        RSG_Kwz,RSG_C11,RSG_C22,RSG_C23,RSG_Krz,RSG_qN,RSG_Kry};

/********************
 *   PROGRAM BODY   *
 ********************/

//ODE system
void fun(double t, DVector &y, DVector &f){
  if ( (fabs(y[1])>0.017) || (fabs(y[5])>0.017) ) {
    cout << endl << y[1] << "\t" << y[5] << endl;
    cout << "Derailment!" << endl;
    exit(1);
  }

  find_contact_forces(t,y,FRONT);
  find_contact_forces(t,y,REAR);

  recalc = false;
  for(int i = 1; i<=N_VAR; i++)
    f[i] = rhs(t,y,i);

  recalc = true;
}

//Jacobi Matrix
```

```
void jac(double t, DVector &y, DMatrix &J) {
  for(int i=1; i <= N_VAR; i++)
    for (int j=1; j <= N_VAR; j++)
      J(i,j)=0;

  for(int i=1; i < (N_VAR-2); i+=2)
    J(i,i+1)=1.0;


  //Analytic part
  J(10,1) = 2*k1/mf;
  J(10,5) = 2*k1/mf;
  J(10,9) = (-4*k1-2*k4)/mf;
  J(10,10) = -2*D2/mf;
  J(10,13) = (2*k4*h2-4*h1*k1)/mf;
  J(10,14) = 2*D2*h2/mf;
  J(10,15) = 2*k4*h3/mf;
  J(10,16) = 2*D2*h3/mf;

  J(12,1) = 2*b*k1/Ifz;
  J(12,3) = 2*d1d1*k2/Ifz;
  J(12,5) = -2*b*k1/Ifz;
  J(12,7) = 2*d1d1*k2/Ifz;
  J(12,11) = -(4*d1d1*k2+k6+4*b*b*k1)/Ifz;
  J(12,12) = -D6/Ifz;

  J(14,1)=2*k1*h1/Ifx;
  J(14,5)=2*k1*h1/Ifx;
  J(14,9)=(-4*k1*h1+2*k4*h2)/Ifx;
  J(14,10)=2*D2*h2/Ifx;
  J(14,13)=(-4*k1*h1*h1-2*k4*h2*h2-2*d2d2*k5-4*d1d1*k3)/Ifx;
  J(14,14)=(-2*D2*h2*h2-2*d2d2*D1)/Ifx;
  J(14,15)=(-2*k4*h2*h3+2*d2d2*k5)/Ifx;
  J(14,16)=(-2*D2*h2*h3+2*d2d2*D1)/Ifx;
  J(14,21)=2*d1d1*k3/Ifx;
  J(14,23)=2*d1d1*k3/Ifx;

  J(16,9)=2*k4*h3/Icx;
  J(16,10)=2*D2*h3/Icx;
  J(16,13)=(2*d2d2*k5 - 2*k4*h3*h2)/Icx;
  J(16,14)=(2*d2d2*D1 - 2*D2*h3*h2)/Icx;
  J(16,15)=(-2*d2d2*k5 - 2*k4*h3*h3 + mrc*g)/Icx;
  J(16,16)=(-2*d2d2*D1 - 2*D2*h3*h3)/Icx;

  J(26,17)=2*k3/mf;
  J(26,19)=2*k3/mf;
  J(26,25)=(-4*k3-2*k5)/mf;
  J(26,26)=-2*D1/mf;
```

```
  J(28,17)=-2*b*k3/Ify;
  J(28,19)=2*b*k3/Ify;
  J(28,27)=-4*b*b*k3/Ify;

  //Numerical part
  fast_num(t,y,J);
}

void fast_num(double t, DVector &y, DMatrix &J) {
  int eqn;
  int v_length;

  //equation 2
  eqn = 2;
  int vars_eq2[] = {1,2,3,4,9,11,13,17,18,21,22,29};
  v_length = 12;
  jac_numerical(t,y,J,eqn,vars_eq2,v_length);

  //equation 4
  eqn = 4;
  int vars_eq4[] = {1,2,3,4,11,17,18,21,22,29};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq4,v_length);

  //equation 6
  eqn = 6;
  int vars_eq6[] = {5,6,7,8,9,11,13,19,20,23,24,30};
  v_length = 12;
  jac_numerical(t,y,J,eqn,vars_eq6,v_length);

  //equation 8
  eqn = 8;
  int vars_eq8[] = {5,6,7,8,11,19,20,23,24,30};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq8,v_length);

  //equation 18
  eqn = 18;
  int vars_eq18[] = {1,2,3,4,17,18,21,22,25,29};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq18,v_length);

  //equation 20
  eqn = 20;
  int vars_eq20[] = {5,6,7,8,19,20,23,24,25,30};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq20,v_length);

  //equation 22
```

```
  eqn = 22;
  int vars_eq22[] = {1,2,3,4,13,17,18,21,22,29};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq22,v_length);

  //equation 24
  eqn = 24;
  int vars_eq24[] = {5,6,7,8,13,19,20,23,24,30};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq24,v_length);

  //equation 29
  eqn = 29;
  int vars_eq29[] = {1,2,3,4,13,17,18,21,22,29};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq29,v_length);

  //equation 30
  eqn = 30;
  int vars_eq30[] = {5,6,7,8,13,19,20,23,24,30};
  v_length = 10;
  jac_numerical(t,y,J,eqn,vars_eq30,v_length);

  // In calculating contact forces
  // the result depends on the following variables:
  //
  // front = {1,2,3,4,17,18,21,22,29}
  // rear = {5,6,7,8,19,20,23,24,30}
}

void jac_numerical(double t, DVector &y, DMatrix &J, int eqn, int vars[],
                   int v_length) {
  DVector y_pos(y);
  DVector y_neg(y);
  double fy_pos,fy_neg;
  double delta = 1e-6;
  for(int i=0; i<v_length; i++){
    y_pos[vars[i]] += delta; //disturb to one side
    fy_pos = rhs(t,y_pos,eqn);
    y_pos[vars[i]] -= delta; //return to center
    y_neg[vars[i]] -= delta; //disturb to the other side
    fy_neg = rhs(t,y_neg,eqn);
    y_neg[vars[i]] += delta; //return to center
    J(eqn,vars[i]) = (fy_pos-fy_neg)/(2.0*delta);
  }
}

/** The following function returns a specific right hand side of the
 *  differential equation system.
```

```
 */
double rhs(double t, DVector &y, int n){
  if(n == 1){
    return y[2];
  }
  else if(n == 2){

    find_contact_forces(t,y,FRONT);

    return (
    c_forces[FRONT][LEFT][FY]+
    c_forces[FRONT][RIGHT][FY]+
    n_forces[FRONT][LEFT][NY]+
    n_forces[FRONT][RIGHT][NY]-
    2*k1*(y[1]-y[9]-b*y[11]-h1*y[13]) -mx*g*phi_SE + mw*v*v/R)/mw;
  }
  else if(n == 3){
    return y[4];
  }
  else if(n == 4){

    find_contact_forces(t,y,FRONT);

    return (
    rsgeodata[FRONT][RIGHT][RSG_Kwy]*(
 c_forces[FRONT][RIGHT][FX]+
 (c_forces[FRONT][RIGHT][FY]+n_forces[FRONT][RIGHT][NY])*y[3])-
    rsgeodata[FRONT][LEFT][RSG_Kwy]*(
 c_forces[FRONT][LEFT][FX]+
 (c_forces[FRONT][LEFT][FY]+n_forces[FRONT][LEFT][NY])*y[3])-
    2*k2*d1d1*(y[3]-y[11]) )/Iwz;
  }
  else if(n == 5){
    return y[6];
  }
  else if(n == 6){

    find_contact_forces(t,y,REAR);

    return (
    c_forces[REAR][LEFT][FY]+
    c_forces[REAR][RIGHT][FY]+
    n_forces[REAR][LEFT][NY]+
    n_forces[REAR][RIGHT][NY]-
    2*k1*(y[5]-y[9]+b*y[11]-h1*y[13]) -mx*g*phi_SE + mw*v*v/R)/mw;
  }
  else if(n == 7){
    return y[8];
  }
```

```
  else if(n == 8){

    find_contact_forces(t,y,REAR);

    return (
    rsgeodata[REAR][RIGHT][RSG_Kwy]*(
c_forces[REAR][RIGHT][FX]+
(c_forces[REAR][RIGHT][FY]+n_forces[REAR][RIGHT][NY])*y[7])-
    rsgeodata[REAR][LEFT][RSG_Kwy]*(
c_forces[REAR][LEFT][FX]+
(c_forces[REAR][LEFT][FY]+n_forces[REAR][LEFT][NY])*y[7])-
    2*k2*d1d1*(y[7]-y[11]) )/Iwz;
  }
  else if(n == 9){
    return y[10];
  }
  else if(n == 10){
    return (
    2*k1*(y[1]+y[5]-2*y[9]-2*h1*y[13])+
    2*k4*(h2*y[13]+h3*y[15]-y[9])+
    2*D2*(h2*y[14]+h3*y[16]-y[10]) -mf*g*phi_SE + mf*v*v/R)/mf;
  }
  else if(n == 11){
    return y[12];
  }
  else if(n == 12){
    return (
    2*d1d1*k2*(y[3]+y[7]-2*y[11])-
    k6*y[11]-
    D6*y[12]+
    2*b*k1*(y[1]-y[5]-2*b*y[11]) )/Ifz;
  }
  else if(n == 13){
    return y[14];
  }
  else if(n == 14){
    return (
    2*k1*h1*(y[1]+y[5]-2*y[9]-2*h1*y[13])+
    2*k4*h2*(y[9]-h2*y[13]-h3*y[15])+
    2*D2*h2*(y[10]-h2*y[14]-h3*y[16])-
    2*d2d2*(k5*(y[13]-y[15])+D1*(y[14]-y[16]))-
    2*d1d1*k3*(2* y[13]-y[21]-y[23]) )/Ifx;
  }
  else if(n == 15){
    return y[16];
  }
  else if(n == 16){
    return (
    -2*d2d2*(k5*(y[15]-y[13])+D1*(y[16]-y[14]))+
```

```
  2*k4*h3*(y[9]-h2*y[13]-h3*y[15])+
  2*D2*h3*(y[10]-h2*y[14]-h3*y[16])+
  h4*mrc*g*(phi_SE+y[15])-
  h4*mrc*v*v/R )/Icx;
}
else if(n == 17){
  return y[18];
}
else if(n == 18){

  find_contact_forces(t,y,FRONT);

  return (
  c_forces[FRONT][LEFT][FZ]+
  c_forces[FRONT][RIGHT][FZ]+
  n_forces[FRONT][LEFT][NZ]+
  n_forces[FRONT][RIGHT][NZ]-
  mx*g+
  2*k3*(y[25]-y[17]) - mw*phi_SE*v*v/R )/mw;
}
else if(n == 19){
  return y[20];
}
else if(n == 20){

  find_contact_forces(t,y,REAR);

  return (
  c_forces[REAR][LEFT][FZ]+
  c_forces[REAR][RIGHT][FZ]+
  n_forces[REAR][LEFT][NZ]+
  n_forces[REAR][RIGHT][NZ]-
  mx*g+
  2*k3*(y[25]-y[19]) -mw*phi_SE*v*v/R )/mw;
}
else if(n == 21){
  return y[22];
}
else if(n == 22){

  find_contact_forces(t,y,FRONT);

  return (
  -rsgeodata[FRONT][RIGHT][RSG_Kwy]*(
c_forces[FRONT][RIGHT][FZ]+
n_forces[FRONT][RIGHT][NZ]-
(c_forces[FRONT][RIGHT][FY]+n_forces[FRONT][RIGHT][NY])*y[21])+
   rsgeodata[FRONT][LEFT][RSG_Kwy]*(
c_forces[FRONT][LEFT][FZ]+
```

```
n_forces[FRONT][LEFT][NZ]-
(c_forces[FRONT][LEFT][FY]+n_forces[FRONT][LEFT][NY])*y[21])-
   2*k3*d1d1*(y[21]-y[13]) )/Iwx;
}
else if(n == 23){
  return y[24];
}
else if(n == 24){

  find_contact_forces(t,y,REAR);

  return (
  -rsgeodata[REAR][RIGHT][RSG_Kwy]*(
c_forces[REAR][RIGHT][FZ]+
n_forces[REAR][RIGHT][NZ]-
(c_forces[REAR][RIGHT][FY]+n_forces[REAR][RIGHT][NY])*y[23])+
   rsgeodata[REAR][LEFT][RSG_Kwy]*(
c_forces[REAR][LEFT][FZ]+
n_forces[REAR][LEFT][NZ]-
(c_forces[REAR][LEFT][FY]+n_forces[REAR][LEFT][NY])*y[23])-
   2*k3*d1d1*(y[23]-y[13]) )/Iwx;
}
else if(n == 25){
  return y[26];
}
else if(n == 26){
  return (
  -2*k3*(2*y[25]-y[17]-y[19])-
  2*k5*y[25]-
  2*D1*y[26] - mf*phi_SE*v*v/R)/mf;
}
else if(n == 27){
  return y[28];
}
else if(n == 28){
  return ( -2*b*k3*(2*b*y[27]+y[17]-y[19]) )/Ify;
}
else if(n == 29){

  find_contact_forces(t,y,FRONT);

  return (
  -rsgeodata[FRONT][RIGHT][RSG_Kwz]*(
c_forces[FRONT][RIGHT][FX]+
(c_forces[FRONT][RIGHT][FY]+n_forces[FRONT][RIGHT][NY])*y[3])-
   rsgeodata[FRONT][LEFT][RSG_Kwz]*(
c_forces[FRONT][LEFT][FX]+
(c_forces[FRONT][LEFT][FY]+n_forces[FRONT][LEFT][NY])*y[3])-
   2*d1d1*k3*y[3]*y[13] )/Iwy;
```

```
  }
  else if(n == 30){

    find_contact_forces(t,y,REAR);

    return (
    -rsgeodata[REAR][RIGHT][RSG_Kwz]*(
  c_forces[REAR][RIGHT][FX]+
  (c_forces[REAR][RIGHT][FY]+n_forces[REAR][RIGHT][NY])*y[7])-
     rsgeodata[REAR][LEFT][RSG_Kwz]*(
  c_forces[REAR][LEFT][FX]+
  (c_forces[REAR][LEFT][FY]+n_forces[REAR][LEFT][NY])*y[7])-
     2*d1d1*k3*y[7]*y[13] )/Iwy;
  }
  else {
    cout << "Unknown equation!" << endl;
  }
}

void find_contact_forces(double t, DVector &y, int wheelset) {
  if(recalc) {
    double N[2];
    double delta[2];
    double cforces[2];
    double Fcreepx;  //with respect to the contact coordinate system
    double Fcreepy; //with respect to the contact coordinate system
    double psi; //wheelset yaw angle
    double phi; //wheelset roll angle

    find_rsgeodata(t,y,wheelset);

    if(wheelset == FRONT) {
      psi = y[3];
      phi = y[21];
    }
    else {
      psi = y[7];
      phi = y[23];
    }

    delta[LEFT] = rsgeodata[wheelset][LEFT][RSG_angle];
    delta[RIGHT] = rsgeodata[wheelset][RIGHT][RSG_angle];

    //Creep forces
    //Left
    SHE(t,y,wheelset,LEFT,cforces);
    Fcreepx = cforces[0];
    Fcreepy = cforces[1];
    c_forces[wheelset][LEFT][FX] = Fcreepx-Fcreepy*psi*cos(delta[LEFT]);
```

```
    c_forces[wheelset][LEFT][FY] = Fcreepx*psi+Fcreepy*cos(delta[LEFT]
                                                          +phi);
    c_forces[wheelset][LEFT][FZ] = Fcreepy*sin(delta[LEFT]+phi);

    //Right
    SHE(t,y,wheelset,RIGHT,cforces);
    Fcreepx = cforces[0];
    Fcreepy = cforces[1];
    c_forces[wheelset][RIGHT][FX] = Fcreepx-Fcreepy*psi*cos(delta[RIGHT]);
    c_forces[wheelset][RIGHT][FY] = Fcreepx*psi+Fcreepy*cos(delta[RIGHT]-
                                                           phi);
    c_forces[wheelset][RIGHT][FZ] = -Fcreepy*sin(delta[RIGHT]-phi);

    //Normal forces
    //Left
    N[LEFT] = rsgeodata[wheelset][LEFT][RSG_N];
    n_forces[wheelset][LEFT][NY] = -N[LEFT]*sin(delta[LEFT]+phi);
    n_forces[wheelset][LEFT][NZ] = N[LEFT]*cos(delta[LEFT]+phi);

    //Right
    N[RIGHT] = rsgeodata[wheelset][RIGHT][RSG_N];
    n_forces[wheelset][RIGHT][NY] = N[RIGHT]*sin(delta[RIGHT]-phi);
    n_forces[wheelset][RIGHT][NZ] = N[RIGHT]*cos(delta[RIGHT]-phi);
  }
}

/** Reads RSGEO data from a file into rsgeo_table
*/
void read_rsgeo() {
  ifstream rsgeofile("pm17mm_b.dat");
  const int numchar=400;
  char buffer[numchar];

  rsgeofile.getline(buffer,numchar);
  for(int i=0; i < N_ROWS_RSGEO; i++)
    for(int j=0; j < N_COLS_RSGEO; j++)
      rsgeofile >> rsgeo_table[i][j];
  rsgeofile.close();
}

/** Interpolation of the RSGEO table, used to find the
 *  exact data for the given lateral displacement.
 */
void find_rsgeodata(double t, DVector &y, int wheelset) {
  double min = -0.01700;
  double max = 0.01700;
  double dd = (max-min)/(N_ROWS_RSGEO-1);
  int index[2];
  double frac[2];
```

```
  double lat_disp[2];

  // In order to reduce the size of rsgeo_table[][] the wheels
  // on the right side are treated as if placed on the left side.
  if(wheelset == FRONT) {
    if(fabs(y[1]) >= max) {
      cout << "Train derailed!" << endl;
      exit(1);
    }
    lat_disp[LEFT] = y[1];
    lat_disp[RIGHT] = -y[1];
  }
  else {
    if(fabs(y[5]) >= max) {
      cout << "Train derailed!" << endl;
      exit(1);
    }
    lat_disp[LEFT] = y[5];
    lat_disp[RIGHT] = -y[5];
  }

  index[LEFT] = int(floor((lat_disp[LEFT]-min)/dd));
  frac[LEFT] = (lat_disp[LEFT]-rsgeo_table[index[LEFT]][0])/dd;
  index[RIGHT] = int(floor((lat_disp[RIGHT]-min)/dd));
  frac[RIGHT] = (lat_disp[RIGHT]-rsgeo_table[index[RIGHT]][0])/dd;

  for(int i = 0; i < N_COLS_RSGEO; i++) {
      rsgeodata[wheelset][LEFT][i] = rsgeo_table[index[LEFT]][i]*
      (1-frac[LEFT])+rsgeo_table[index[LEFT]+1][i]*frac[LEFT];
      rsgeodata[wheelset][RIGHT][i] = rsgeo_table[index[RIGHT]][i]*
      (1-frac[RIGHT])+rsgeo_table[index[RIGHT]+1][i]*frac[RIGHT];
  }

  dynamic_update(t,y,wheelset);
}

void dynamic_update(double t, DVector &y, int wheelset) {
  double y_lat,z_ver,phi;

  if(wheelset == FRONT) {
   y_lat = y[1];
z_ver = y[17];
phi = y[21];
  }
  else {
   y_lat = y[5];
z_ver = y[19];
phi = y[23];
  }
```

```
  dynamic_aux(
   rsgeodata[wheelset][LEFT][RSG_Kwz],
   rsgeodata[wheelset][LEFT][RSG_Kwy],
   rsgeodata[wheelset][LEFT][RSG_Kry],
   rsgeodata[wheelset][LEFT][RSG_Krz],
   rsgeodata[wheelset][LEFT][RSG_angle],
   y_lat,
   z_ver,
   phi,
   wheelset,
   LEFT);

  dynamic_aux(
   rsgeodata[wheelset][RIGHT][RSG_Kwz],
   rsgeodata[wheelset][RIGHT][RSG_Kwy],
   rsgeodata[wheelset][RIGHT][RSG_Kry],
   rsgeodata[wheelset][RIGHT][RSG_Krz],
   rsgeodata[wheelset][RIGHT][RSG_angle],
   y_lat,
   z_ver,
   phi,
   wheelset,
   RIGHT);
}

void dynamic_aux(
 double radius,
 double aw,
 double at,
 double zt,
 double delta,
 double y_lat,
 double z_ver,
 double phi,
 int wheelset,
 int wheel) {

  double dq,N,N0,N3,q0;

  if(wheel == LEFT) dq = -(at - y_lat - aw - phi*radius)*sin(delta+phi)+
                          (-z_ver - phi*aw)*cos(delta+phi);
  else dq = (-at - y_lat + aw - phi*radius)*sin(delta-phi)+(-z_ver +
             phi*aw)*cos(delta-phi);

  N0 = rsgeodata[wheelset][wheel][RSG_N];
  q0 = rsgeodata[wheelset][wheel][RSG_qN];
  N = N0*pow(1+dq/q0,1.5); if(isnan(N)) N = 0;
  N3 = pow(N/N0,1.0/3.0);
```

```
  rsgeodata[wheelset][wheel][RSG_a] *= N3;
  rsgeodata[wheelset][wheel][RSG_b] *= N3;
  rsgeodata[wheelset][wheel][RSG_N] = N;
}

void SHE(double t, DVector &y, int wheelset, int wheel, double* out) {
  double norm_F_tau; //2-norm of the tangential force in the contact plane
  double epsilon; //Reduction coefficient
  double Fx_val; //Creep force wrt the contact coordinate system
  double Fy_val; //Creep force wrt the contact coordinate system
  double cp_a,cp_b; //Semi axes of the contact ellipse
  double C11,C22,C23; //Kalker's creepage coefficients
  double xi_x,xi_y,xi_s; //Lateral, Longitudinal and spin creep
  double radius; //Rolling radius of left and right wheel
  double aw; //distance to contact point
  double delta; //Angle between the wheel axle and the contact plane
  double N; //Normal force
  double phi;

  N = rsgeodata[wheelset][wheel][RSG_N];
  cp_a = rsgeodata[wheelset][wheel][RSG_a];
  cp_b = rsgeodata[wheelset][wheel][RSG_b];
  C11 = rsgeodata[wheelset][wheel][RSG_C11];
  C22 = rsgeodata[wheelset][wheel][RSG_C22];
  C23 = rsgeodata[wheelset][wheel][RSG_C23];
  radius = rsgeodata[wheelset][wheel][RSG_Kwz];
  aw = rsgeodata[wheelset][wheel][RSG_Kwy];
  delta = rsgeodata[wheelset][wheel][RSG_angle];

  if(wheelset == FRONT) {
    if(wheel == LEFT) {
      xi_x = xi_flx(y[3],y[4],y[22],y[2],y[29],radius,aw);
      xi_y = xi_fly(y[3],y[21],y[22],y[2],y[18],radius,aw,delta);
      xi_s = xi_fls(y[3],y[4],y[22],y[29],delta);
    }
    else {
      xi_x = xi_frx(y[3],y[4],y[22],y[2],y[29],radius,aw);
      xi_y = xi_fry(y[3],y[21],y[22],y[2],y[18],radius,aw,delta);
      xi_s = xi_frs(y[3],y[4],y[22],y[29],delta);
    }
  }
  else {
    if(wheel == LEFT) {
      xi_x = xi_rlx(y[7],y[8],y[24],y[6],y[30],radius,aw);
      xi_y = xi_rly(y[7],y[23],y[24],y[6],y[20],radius,aw,delta);
      xi_s = xi_rls(y[7],y[8],y[24],y[30],delta);
    }
    else {
      xi_x = xi_rrx(y[7],y[8],y[24],y[6],y[30],radius,aw);
```

```
      xi_y = xi_rry(y[7],y[23],y[24],y[6],y[20],radius,aw,delta);
      xi_s = xi_rrs(y[7],y[8],y[24],y[30],delta);
    }
  }

  if(N == 0) {
    out[0] = 0;
    out[1] = 0;
    return;
  }

  Fx_val = Fx(cp_a,cp_b,C11,xi_x);
  Fy_val = Fy(cp_a,cp_b,C22,C23,xi_y,xi_s);

  double norm_F_tau_ping = sqrt(pow(Fx_val,2.0)+pow(Fy_val,2.0));
  double u = (norm_F_tau_ping/(mu*N));

  if(u<3) {
    norm_F_tau = mu*N*(u - 1.0/3.0*pow(u,2.0)+1.0/27.0*pow(u,3.0));
  }
  else {
    norm_F_tau = mu*N;
  }

  epsilon = norm_F_tau/norm_F_tau_ping;
  out[0] = epsilon*Fx_val;
  out[1] = epsilon*Fy_val;
}

inline double xi_flx(double psi, double psidot, double phidot,
                     double ydot, double beta, double rl, double al) {
  Omega_0 = v/r0;
  return (v + (psi+b/R)*ydot + rl*((psi+b/R)*phidot - (Omega_0+beta)) -
          al*psidot + a*v*cos(phi_SE)/R )/v;
}

inline double xi_frx(double psi, double psidot, double phidot,
                     double ydot, double beta, double rr, double ar) {
  Omega_0 = v/r0;
  return (v + (psi+b/R)*ydot + rr*((psi+b/R)*phidot - (Omega_0+beta)) +
          ar*psidot - a*v*cos(phi_SE)/R )/v;
}

inline double xi_fly(double psi, double phi, double phidot, double ydot,
                     double zdot, double rl, double al, double deltal) {
  return ((-v*(psi+b/R)*(1+a*cos(phi_SE)/R)+phi*zdot+ydot+phidot*rl)*
           cos(deltal) + (-phi*ydot+zdot+al*phidot)*sin(deltal))/v;
}
```

```
inline double xi_fry(double psi, double phi, double phidot, double ydot,
                     double zdot, double rr, double ar, double deltar) {
  return ((-v*(psi+b/R)*(1-a*cos(phi_SE)/R)+phi*zdot+ydot+phidot*rr)*
          cos(deltar) - (-phi*ydot+zdot-ar*phidot)*sin(deltar))/v;
}

inline double xi_rlx(double psi, double psidot, double phidot,
                     double ydot, double beta, double rl, double al) {
  Omega_0 = v/r0;
  return (v + (psi-b/R)*ydot + rl*((psi-b/R)*phidot - (Omega_0+beta)) -
          al*psidot + a*v*cos(phi_SE)/R )/v;
}

inline double xi_rrx(double psi, double psidot, double phidot,
                     double ydot, double beta, double rr, double ar) {
  Omega_0 = v/r0;
  return (v + (psi-b/R)*ydot + rr*((psi-b/R)*phidot - (Omega_0+beta)) +
          ar*psidot - a*v*cos(phi_SE)/R )/v;
}

inline double xi_rly(double psi, double phi, double phidot, double ydot,
                     double zdot, double rl, double al, double deltal) {
  return ((-v*(psi-b/R)*(1+a*cos(phi_SE)/R)+phi*zdot+ydot+phidot*rl)*
          cos(deltal) + (-phi*ydot+zdot+al*phidot)*sin(deltal))/v;
}

inline double xi_rry(double psi, double phi, double phidot, double ydot,
                     double zdot, double rr, double ar, double deltar) {
  return ((-v*(psi-b/R)*(1-a*cos(phi_SE)/R)+phi*zdot+ydot+phidot*rr)*
           cos(deltar) - (-phi*ydot+zdot-ar*phidot)*sin(deltar))/v;
}

inline double xi_fls(double psi, double psidot, double phidot,
                     double beta, double deltal) {
  Omega_0 = v/r0;
  return (-(Omega_0+beta-(psi+b/R)*phidot)*sin(deltal)+psidot*
           cos(deltal))/v;
}

inline double xi_frs(double psi, double psidot, double phidot,
                     double beta, double deltar) {
  Omega_0 = v/r0;
  return ((Omega_0+beta-(psi+b/R)*phidot)*sin(deltar)+psidot*
           cos(deltar))/v;
}

inline double xi_rls(double psi, double psidot, double phidot,
                     double beta, double deltal) {
  Omega_0 = v/r0;
```

```
  return (-(Omega_0+beta-(psi-b/R)*phidot)*sin(deltal)+psidot*
           cos(deltal))/v;
}

inline double xi_rrs(double psi, double psidot, double phidot,
                     double beta, double deltar) {
  Omega_0 = v/r0;
  return ((Omega_0+beta-(psi-b/R)*phidot)*sin(deltar)+psidot*
          cos(deltar))/v;
}

inline double Fx(double cp_a, double cp_b, double C11, double xi_x) {
  return (-cp_a*cp_b*G*C11*xi_x);
}

inline double Fy(double cp_a, double cp_b, double C22, double C23,
                 double xi_y, double xi_s) {
  return (-cp_a*cp_b*G*(C22*xi_y+sqrt(cp_a*cp_b)*C23*xi_s));
}

void print_jac(double t,DVector &y, int number) {
  char filename[120];
  sprintf(filename,"jacobi_%i.dat",number);
  ofstream jacobifile(filename); //Output file
  DMatrix J(30,30);

  jac(t,y,J);

  for(int i=1;i<=30;i++) {
    for(int j=1;j<=30;j++) {
      jacobifile << J(i,j) << " ";
    }
    jacobifile << "\n";
  }

  jacobifile.close();
}

void print_rsgeo() {
  ofstream rsgeo_tab("rsgeotable");

  for(int i=0; i < N_ROWS_RSGEO; i++) {
    for(int j=0; j < N_COLS_RSGEO; j++) {
      rsgeo_tab << rsgeo_table[i][j] << " ";
    }
    rsgeo_tab << "\n";
  }
  rsgeo_tab.close();
}
```

```
void toFile(double t,DVector &y) {
  //Solution
  sol << t << " "
      << y[1] << " " << y[2] << " " << y[3] << " " << y[4] << " " << y[5]
      << " "
      << y[6] << " " << y[7] << " " << y[8] << " " << y[9] << " " << y[10]
      << " "
      << y[11] << " " << y[12] << " " << y[13] << " " << y[14] << " "
      << y[15] << " "
      << y[16] << " " << y[17] << " " << y[18] << " " << y[19] << " "
      << y[20] << " "
      << y[21] << " " << y[22] << " " << y[23] << " " << y[24] << " "
      << y[25] << " "
      << y[26] << " " << y[27] << " " << y[28] << " " << y[29] << " "
      << y[30] << "\n";

  //Creep forces
  forces << t << " "
 << c_forces[FRONT][LEFT][FX] << " " << c_forces[FRONT][LEFT][FY]
         << " " << c_forces[FRONT][LEFT][FZ] << " "
 <<  c_forces[FRONT][RIGHT][FX] << " " << c_forces[FRONT][RIGHT][FY]
         << " " << c_forces[FRONT][RIGHT][FZ] << " "
 <<  c_forces[REAR][LEFT][FX] << " " << c_forces[REAR][LEFT][FY]
         << " " << c_forces[REAR][LEFT][FZ] << " "
 <<  c_forces[REAR][RIGHT][FX] << " " << c_forces[REAR][RIGHT][FY]
         << " " << c_forces[REAR][RIGHT][FZ] << " ";

  //Normal forces
  forces << n_forces[FRONT][LEFT][NY] << " " << n_forces[FRONT][LEFT][NZ]
         << " "
 << n_forces[FRONT][RIGHT][NY] << " " << n_forces[FRONT][RIGHT][NZ]
         << " "
 << n_forces[REAR][LEFT][NY] << " " << n_forces[REAR][LEFT][NZ]
         << " "
 << n_forces[REAR][RIGHT][NY] << " " << n_forces[REAR][RIGHT][NZ]
         << "\n";
}

void solve() {
  DVector y(30),T(2);

  // Initializing the Sdirk instance MyInstance
  Sdirk *MyInstance;
  MyInstance = new Sdirk(1e-8, N_VAR, &fun, &jac, SC_PI);

  cout << "Reading RSGEO table ...";
  read_rsgeo();
  cout << " done."<<endl;
```

```
// Initial values
//for(int i=1; i<=N_VAR; i++) y[i] = 0.0;
use_last(y);

double t_lo = 0.0, t_hi = 2.0; //Integration interval
int m = (t_hi-t_lo)*1000; //Number of outputs
double dt = (t_hi-t_lo)/m; //Distance between outputs
double h=0.001; //Initial steplength
long Nstep=0; //Number of steps used in each integration
int i;
char output[128];

//display information
cout<<"****************************************"<<endl;
cout<<"           SOLVER INFORMATION           "<<endl;
cout<<"****************************************"<<endl;
cout<<"  Time span = ["<<t_lo<<", "<<t_hi<<"]"<<endl;
cout<<"  outputs   = "<<m<<endl;
cout<<"  dt        = "<<dt<<endl;
cout<<"  initial h = "<<h<<endl;
cout<<"****************************************"<<endl;
//end display information

//Write initial state
toFile(t_lo,y);
for (i=0; i<m; i++) {
  T[1] = t_lo+i*dt; //Start time of integration step
  T[2] = T[1]+dt; //End time of integration step

  for(int j = 0; j < 130; j++)
    cout << "\b";

  sprintf(output,"(t,v,R,phi_SE) = (%g,%g,%g,%g)",T[1],v,R,phi_SE*
          180/M_PI);
  cout << output << flush;

  //linear modifications (overrides those in main)
  v = 35 + 0.1*T[1];
  R = 1500 - 20*T[1];
  phi_SE = (2 + 0.2*T[1])*M_PI/180;

  MyInstance->Integrate(T,h,y,Nstep); //Integration

  if(i%10==0) //use only if to avoid enormous files
    toFile(T[2],y);
}
cout << endl;
MyInstance->~Sdirk(); //Free the instance
```

```
  latyaw << y[1] << " " << y[5] << " " << y[9] << " " << y[3] << " "
         << y[7] << " " << y[11] << "\n";
  output_last(y);
  output_last_with_y(y);
  print_jac(T[1],y,1);
}

int main() {
  cout << "Starting program !!" << endl;

  //Set parameters
  v = 50;
  R = 5000;
  phi_SE = 2*M_PI/180;

  solve();
  //latyaw_analyse();

  sol.close(); //Close the solution file
  forces.close(); //Close the forces file
  latyaw.close(); //Close the lat yaw file
  cout << "Bye ..." << endl;
  return 0;
}

void latyaw_analyse() {
  for(int i=0;i<=40;i++) {
    cout << i << endl;
    R = 1200+i*20;
    solve();
  }
}

void output_last(DVector &y){
  ofstream yfile("last.dat");
  for(int i = 1; i <= 30; i++)
    yfile << y[i] << "\n";
  yfile.close();
}

void output_last_with_y(DVector &y){
  ofstream yfile("yfinal.dat");
  for(int i = 1; i <= 30; i++)
    yfile << "y[" << i << "] = "<< y[i] << ";";
  yfile.close();
}

void use_last(DVector &y){
```

```
  ifstream yfile("last.dat");
  for(int i = 1; i <= 30; i++)
    yfile >> y[i];
  yfile.close();
}

/* Transformation

 q1     = y1
 q1dot  = y2
 q2 = y3
 q2dot  = y4
 q3 = y5
 q3dot  = y6
 q4 = y7
 q4dot  = y8
 q5 = y9
 q5dot  = y10
 q6 = y11
 q6dot  = y12
 q7 = y13
 q7dot  = y14
 q8 = y15
 q8dot  = y16
 q9     = y17
 q9dot  = y18
 q10    = y19
 q10dot = y20
 q11    = y21
 q11dot = y22
 q12    = y23
 q12dot = y24
 q13    = y25
 q13dot = y26
 q14    = y27
 q14dot = y28
 betaf  = y29
 betar  = y30

 y[1]  : Front wheelset lateral
 y[3]  : Front wheelset yaw      (psi)
 y[5]  : Rear wheelset lateral
 y[7]  : Rear wheelset yaw       (psi)
 y[9]  : Bogie frame lateral
 y[11] : Bogie frame yaw
 y[13] : Bogie frame roll
 y[15] : Car roll
 y[17] : Front wheelset vertical
 y[19] : Rear wheelset vertical
```

```
 y[21] : Front wheelset roll     (phi)
 y[23] : Rear wheelset roll      (phi)
 y[25] : Bogie frame vertical
 y[27] : Bogie frame pitch
 y[29] : Rolling constraint beta1 (front wheelset)
 y[30] : Rolling constraint beta2 (rear wheelset)
*/
```

E.2 Makefile

```
# Makefile for implementation linking to libsdirk.a
#
# Source to make: model.cc
TARGET = model

# Libraries and headers
LIB = ../../../sdirk/lib
INC = ../../../sdirk/include

# Compiler and options
CC = g++
CCOPTS = -Wall -I$(INC) -c -w

# Rules
all : $(TARGET)

$(TARGET): $(TARGET).o
$(CC) -L$(LIB) -o $(TARGET) $(TARGET).o -lsdirk -lm
$(TARGET).o: $(TARGET).cc
$(CC) $(CCOPTS) $(TARGET).cc


clean :
rm -f $(TARGET) *.o model sol.dat forces.dat *~
```

Appendix F

MATLAB Source Code

F.1 plotFORCES.m

```
function plotFORCES(n,FORCES,fig,parm)

labelize = 1;

if(nargin == 2)
  close all
  figure(n)
  clf
  plot(FORCES(:,1),FORCES(:,n),'r.')
  grid
elseif(nargin == 4)
  labelize = 0;
  figure(fig)
  hold on
  plot(FORCES(:,1),FORCES(:,n),parm)
  return
else
  disp('Input Error')
  return
end


if(labelize)
  switch(n)
   case 1,
    clf
    plot(diff(FORCES(:,1)),'.');
    title('Time Step Size At Step Number');
    xlabel('integration step');
    ylabel('step size');
   case 2,
```

```
 title('Creep Force - Front Left F_x')
 xlabel('time (s)')
 ylabel('force (N)')
case 3,
 title('Creep Force - Front Left F_y')
 xlabel('time (s)')
 ylabel('force (N)')
case 4,
 title('Creep Force - Front Left F_z')
 xlabel('time (s)')
 ylabel('force (N)')
case 5,
 title('Creep Force - Front Right F_x')
 xlabel('time (s)')
 ylabel('force (N)')
case 6,
 title('Creep Force - Front Right F_y')
 xlabel('time (s)')
 ylabel('force (N)')
case 7,
 title('Creep Force - Front Right F_z')
 xlabel('time (s)')
 ylabel('force (N)')
case 8,
 title('Creep Force - Rear Left F_x')
 xlabel('time (s)')
 ylabel('force (N)')
case 9,
 title('Creep Force - Rear Left F_y')
 xlabel('time (s)')
 ylabel('force (N)')
case 10,
 title('Creep Force - Rear Left F_z')
 xlabel('time (s)')
 ylabel('force (N)')
case 11,
 title('Creep Force - Rear Right F_x')
 xlabel('time (s)')
 ylabel('force (N)')
case 12,
 title('Creep Force - Rear Right F_y')
 xlabel('time (s)')
 ylabel('force (N)')
case 13,
 title('Creep Force - Rear Right F_z')
 xlabel('time (s)')
 ylabel('force (N)')
case 14,
 title('Normal Force - Front Left N_y')
```

```
    xlabel('time (s)')
    ylabel('force (N)')
   case 15,
    title('Normal Force - Front Left N_z')
    xlabel('time (s)')
    ylabel('force (N)')
   case 16,
    title('Normal Force - Front Right N_y')
    xlabel('time (s)')
    ylabel('force (N)')
   case 17,
    title('Normal Force - Front Right N_z')
    xlabel('time (s)')
    ylabel('force (N)')
   case 18,
    title('Normal Force - Rear Left N_y')
    xlabel('time (s)')
    ylabel('force (N)')
   case 19,
    title('Normal Force - Rear Left N_z')
    xlabel('time (s)')
    ylabel('force (N)')
   case 20,
    title('Normal Force - Rear Right N_y')
    xlabel('time (s)')
    ylabel('force (N)')
   case 21,
    title('Normal Force - Rear Right N_z')
    xlabel('time (s)')
    ylabel('force (N)')
  end
end
```

F.2 plotSOL.m

```
function plotSOL(n,SOL,fig,parm)

labelize = 1;

if(nargin == 2)
  close all
  figure(n)
  clf
  plot(SOL(:,1),SOL(:,n),'r.')
  grid
```

```
elseif(nargin == 4)
  labelize = 0;
  figure(fig)
  hold on
  plot(SOL(:,1),SOL(:,n),parm)
  return
else
  disp('Input Error')
  return
end


if(labelize)
  switch(n)
   case 1,
    clf
    plot(diff(SOL(:,1)),'.');
    title('Time Step Size At Step Number');
    xlabel('integration step');
    ylabel('step size');
   case 2,
    title('Front Wheelset Lateral Displacement q_1');
    xlabel('time (s)');
    ylabel('displacement (m)');
   case 3,
    title('Front Wheelset Lateral Velocity q''_1');
    xlabel('time (s)');
    ylabel('velocity (m s^{-1})');
   case 4,
    title('Front Wheelset Yaw q_2');
    xlabel('time (s)');
    ylabel('yaw angle (rad)');
   case 5,
    title('Front Wheelset Yaw Velocity q''_2');
    xlabel('time (s)');
    ylabel('yaw angular velocity (rad s^{-1})');
   case 6,
    title('Rear Wheelset Lateral Displacement q_3');
    xlabel('time (s)');
    ylabel('displacement (m)');
   case 7,
    title('Rear Wheelset Lateral Velocity q''_3');
    xlabel('time (s)');
    ylabel('velocity (m s^{-1})');
   case 8,
    title('Rear Wheelset Yaw q_4');
    xlabel('time (s)');
    ylabel('yaw angle (rad)');
   case 9,
```

```
 title('Rear Wheelset Yaw Velocity q''_4');
 xlabel('time (s)');
 ylabel('yaw angular velocity (rad s^{-1})');
case 10,
 title('Bogie Frame Lateral Displacement q_5');
 xlabel('time (s)');
 ylabel('displacement (m)');
case 11,
 title('Bogie Frame Lateral Velocity q''_5');
 xlabel('time (s)');
 ylabel('velocity (m s^{-1})');
case 12,
 title('Bogie Frame Yaw q_6');
 xlabel('time (s)');
 ylabel('yaw angle (rad)');
case 13,
 title('Bogie Frame Yaw Velocity q''_6');
 xlabel('time (s)');
 ylabel('yaw angular velocity (rad s^{-1})');
case 14,
 title('Bogie Frame Roll q_7');
 xlabel('time (s)');
 ylabel('roll angle (rad)');
case 15,
 title('Bogie Frame Roll Velocity q''_7');
 xlabel('time (s)');
 ylabel('roll angular velocity (rad s^{-1})');
case 16,
 title('Rail Car Roll q_8');
 xlabel('time (s)');
 ylabel('roll angle (rad)');
case 17,
 title('Rail Car Roll Velocity q''_8');
 xlabel('time (s)');
 ylabel('roll angular velocity (rad s^{-1})');
case 18,
 title('Front Wheelset Vertical Displacement q_9');
 xlabel('time (s)');
 ylabel('displacement (m)');
case 19,
 title('Front Wheelset Vertical Velocity q''_9');
 xlabel('time (s)');
 ylabel('velocity (m s^{-1})');
case 20,
 title('Rear Wheelset Vertical Displacement q_{10}');
 xlabel('time (s)');
 ylabel('displacement (m)');
case 21,
 title('Rear Wheelset Vertical Velocity q''_{10}');
```

```
    xlabel('time (s)');
    ylabel('velocity (m s^{-1})');
   case 22,
    title('Front Wheelset Roll q_{11}');
    xlabel('time (s)');
    ylabel('roll angle (rad)');
   case 23,
    title('Front Wheelset Roll Velocity q''_{11}');
    xlabel('time (s)');
    ylabel('roll angular velocity (rad s^{-1})');
   case 24,
    title('Rear Wheelset Roll q_{12}');
    xlabel('time (s)');
    ylabel('roll angle (rad)');
   case 25,
    title('Rear Wheelset Roll Velocity q''_{12}');
    xlabel('time (s)');
    ylabel('roll angular velocity (rad s^{-1})');
   case 26,
    title('Bogie Frame Vertical Displacement q_{13}');
    xlabel('time (s)');
    ylabel('displacement (m)');
   case 27,
    title('Bogie Frame Vertical Velocity q''_{13}');
    xlabel('time (s)');
    ylabel('velocity (m s^{-1})');
   case 28,
    title('Frame Pitch q_{14}');
    xlabel('time (s)');
    ylabel('pitch angle (rad)');
   case 29,
    title('Frame Pitch Velocity q''_{14}');
    xlabel('time (s)');
    ylabel('pitch angular velocity (rad s^{-1})');
   case 30,
    title('Front Wheelset Roll Constraint \beta_1');
    xlabel('time (s)');
    ylabel('(rad s^{-1})');
   case 31,
    title('Rear Wheelset Roll Constraint \beta_2');
    xlabel('time (s)');
    ylabel('(rad s^{-1})');
  end
end
```

F.3 visualize.m

```
function [SOL,FORCES] =  visualize(R,loaded,figures,SOL,FORCES)

if(nargin == 0)
  %R = 1e30;
  loaded = 0;
  figures = 0;
  SOL = 0;
  FORCES = 0;
end

if(nargin == 1)
  loaded = 0;
  figures = 0;
  SOL = 0;
  FORCES = 0;
end
% MATLAB file to visualize output from the c++ solver
% -loaded determines if data has been loaded from the files.

%if(~exist('loaded'))
%  loaded = 0;
%end

if (figures == 0)
  figures = [1,2,3,4,5,6,7,8,9,10,11];
end

if(~loaded)
  disp('loading...')
  SOL = load('sol.dat');
  FORCES = load('forces.dat');
  loaded = 1;
  disp('  done.')
end

%close all

% The SOL Structure
% Column ! Data
% 1      ! Time
% 2      ! FW Lateral
% 3      ! d(2)/dt
% 4      ! FW Yaw
% 5      ! d(4)/dt
% 6      ! RW Lateral
% 7      ! d(6)/dt
% 8      ! RW Yaw
```

```
% 9      ! d(8)/dt
% 10     ! Frame Lateral
% 11     ! d(10)/dt
% 12     ! Frame Yaw
% 13     ! d(12)/dt
% 14     ! Frame Roll
% 15     ! d(14)/dt
% 16     ! Car Roll
% 17     ! d(16)/dt
% 18     ! FW Vertical
% 19     ! d(18)/dt
% 20     ! RW Vertical
% 21     ! d(20)/dt
% 22     ! FW Roll
% 23     ! d(22)/dt
% 24     ! RW Roll
% 25     ! d(24)/dt
% 26     ! Frame Vertical
% 27     ! d(26)/dt
% 28     ! Frame Pitch
% 29     ! d(28)/dt
% 30     ! FW Roll Constraint
% 31     ! RW Roll Constraint

%Plot Lateral Displacements

if(ismember(1,figures))
  figure(1)
  clf
  title('Lateral Displacements')
  plotSOL(2,SOL,1,'r-')
  plotSOL(6,SOL,1,'b-')
  plotSOL(10,SOL,1,'g-')
  legend('Front Wheelset','Rear Wheelset','Bogie Frame')
  xlabel('time (s)')
  ylabel('displacement (m)')
end

if(ismember(2,figures))
%Plot Vertical Displacements
figure(2)
clf
title('Vertical Displacements')
plotSOL(18,SOL,2,'r-')
plotSOL(20,SOL,2,'b-')
plotSOL(26,SOL,2,'g-')
legend('Front Wheelset','Rear Wheelset','Bogie Frame')
xlabel('time (s)')
ylabel('displacement (m)')
```

```
end

if(ismember(3,figures))
%Plot Roll
figure(3)
clf
title('Rolls')
plotSOL(14,SOL,3,'r-')
plotSOL(16,SOL,3,'b-')
plotSOL(22,SOL,3,'g-')
plotSOL(24,SOL,3,'c-')
legend('Bogie Frame','Car','Front Wheelset','Rear Wheelset')
xlabel('time (s)')
ylabel('roll angle (rad)')
end

if(ismember(4,figures))
%Plot Yaw
figure(4)
clf
title('Yaws')


plotSOL(4,SOL,4,'r-')
plotSOL(8,SOL,4,'b-')
plotSOL(12,SOL,4,'g-')
%expected radial configuration
if(exist('R'))
  b = 1.074;
  plot([SOL(1,1) SOL(end,1)],[b/R b/R], 'k--')
  plot([SOL(1,1) SOL(end,1)],[-b/R -b/R],'k--')
  legend('Front Wheelset','Rear Wheelset','Bogie Frame','Expected');
else
  legend('Front Wheelset','Rear Wheelset','Bogie Frame');
end

xlabel('time (s)')
ylabel('yaw angle (rad)')
end

if(ismember(5,figures))
%Plot Pitch
figure(5)
clf
title('Bogie Frame Pitch')
plotSOL(28,SOL,5,'r-')
xlabel('time (s)')
ylabel('pitch angle (rad)')
end
```

```
if(ismember(6,figures))
%Plot Wheel Roll Constraints
figure(6)
clf
title('Wheelset Roll Constraints')
plotSOL(30,SOL,6,'r-')
plotSOL(31,SOL,6,'b-')
legend('Front Wheelset','Rear Wheelset')
xlabel('time (s)')
ylabel('roll constraint angle (rad)')
end

if(ismember(7,figures))
%Plot Creep Force F_x
figure(7)
clf
title('Creep Forces F_x')
plotFORCES(2,FORCES,7,'r-')
plotFORCES(5,FORCES,7,'b-')
plotFORCES(8,FORCES,7,'g-')
plotFORCES(11,FORCES,7,'c-')
legend('Front Left','Front Right','Rear Left','Rear Right')
xlabel('time (s)')
ylabel('forces (N)')
end

if(ismember(8,figures))
%Plot Creep Force F_y
figure(8)
clf
title('Creep Forces F_y')
plotFORCES(3,FORCES,8,'r-')
plotFORCES(6,FORCES,8,'b-')
plotFORCES(9,FORCES,8,'g-')
plotFORCES(12,FORCES,8,'c-')
legend('Front Left','Front Right','Rear Left','Rear Right')
xlabel('time (s)')
ylabel('forces (N)')
end

if(ismember(9,figures))
%Plot Creep Force F_z
figure(9)
clf
title('Creep Forces F_z')
plotFORCES(4,FORCES,9,'r-')
plotFORCES(7,FORCES,9,'b-')
plotFORCES(10,FORCES,9,'g-')
```

```
plotFORCES(13,FORCES,9,'c-')
legend('Front Left','Front Right','Rear Left','Rear Right')
xlabel('time (s)')
ylabel('forces (N)')
end

if(ismember(10,figures))
%Plot Normal Force N_y
figure(10)
clf
title('Normal Forces N_y')
plotFORCES(14,FORCES,10,'r-')
plotFORCES(16,FORCES,10,'b-')
plotFORCES(18,FORCES,10,'g-')
plotFORCES(20,FORCES,10,'c-')
legend('Front Left','Front Right','Rear Left','Rear Right')
xlabel('time (s)')
ylabel('forces (N)')
end

if(ismember(11,figures))
%Plot Normal Force N_z
figure(11)
clf
title('Normal Forces N_z')
plotFORCES(15,FORCES,11,'r-')
plotFORCES(17,FORCES,11,'b-')
plotFORCES(19,FORCES,11,'g-')
plotFORCES(21,FORCES,11,'c-')
legend('Front Left','Front Right','Rear Left','Rear Right')
xlabel('time (s)')
ylabel('forces (N)')
end

figure(12)
clf
t = linspace(0,10,10000);
title('The Search for the Non-linear Critical Velocity');
hold on;
plot(40+0.5*SOL(:,1),SOL(:,2),'b-');
plot(40+0.5*SOL(:,1),SOL(:,6),'r-');
legend('Front Wheelset','Rear Wheelset')
xlabel('velocity (m/s)')
ylabel('displacement (m)')
```

Bibliography

[1] Lasse Engbo Christiansen. The dynamics of a railway vehicle on a disturbed track. Master's thesis, Technical University of Denmark, June 2001.

[2] Jens Christian Jensen. *Teorestiske og eksperimentelle dynamiske undersøgelser af jernbanekøretøjer.* PhD thesis, Technical University of Denmark, 1995.

[3] Klaus Knothe Jerzy Kisilowski, editor. *Advanced Railway Vehicle System Dynamics.* Wydawnictwa Naukowo-Techniczne, 1991.

[4] Steven H. Strogatz. *Nonlinear Dynamics and Chaos.* Perseus Books, Cambridge Massachusetts, 1994.

[5] Henrik Thillman. Non-linear railway vehicle dynamics. Master's thesis, Technical University of Denmark, 2000.

[6] Hans True. Bifurcation problems in railway vehicle dynamics. In Seydel Küpper and Troger, editors, *Bifurcation: Analysis, Algorithms, Applications.* University of Dortmund, August 1986.