The approximation of contact conditions in the dynamical simulation of wheel-rail systems

Martin Arnold* Helmuth Netter†

Abstract

Models for the geometrical contact of wheel and rail are a basic component of multibody system (MBS) models for wheel-rail systems. Approximations are used to get sufficiently differentiable contact conditions that can be evaluated efficiently. We discuss an approximation that is essentially based on polynomial 2D-tensorproduct splines and minimizes a functional that combines a weighted least squares approximation of a rigid contact model with a smoothing term. An efficient algorithm to compute the tensorproduct spline is developed, the parallelization on a cluster of workstations is discussed. We report on results of both the sequential and the parallel algorithm and give simulation results for a rigid wheelset on a straight track.

1 Introduction

Dynamical simulation is a more and more essential part in the design and analysis processes of advanced railway vehicles. Within simulation packages the dynamical behaviour of wheel-rail systems has to be analysed for large sets of parameters. This motivates the development of robust tools for the efficient dynamical simulation of wheel-rail systems. In this field the mechanical multibody system (MBS) approach is well established: The motion of the wheel-rail system is described by the Euler–Lagrange equations

$$\begin{aligned} M(q)\ddot{q} &= f(q,\dot{q},\lambda,t) - G^T(q,t)\lambda \\ 0 &= g(q,t) \end{aligned} \tag{1}$$

in the position coordinates q ([14]). Here $M(q)$ denotes the symmetric, positive definite mass matrix and $f(q,\dot{q},\lambda,t)$ the applied forces and friction forces, which depend on the constraint forces $-G^T(q,t)\lambda$. The constraints $g(q,t)=0$ are coupled to the dynamical equations by $G(q,t) := g_q(q,t)$ and the Lagrangian multipliers λ. They guarantee that the wheels are always in contact with the rails (*contact conditions*). Consider as an example a rigid wheelset that is described by 6 second order differential equations (1) (corresponding to the 6 degrees of freedom) and 2 contact conditions $g_1(q,t) = g_2(q,t) = 0$ such that neither the left nor the right wheel leaves the rails (Examples 4 and 5).

Eqs. (1) form a differential-algebraic system of index 3. Theory, numerical analysis and software for (1) are well developed (see e. g. [2] and the references therein). We do not focus on this point in the present paper and note only that (1) contains inherently derivatives of

*University of Rostock, Department of Mathematics, Postfach, D – 18051 Rostock, Germany, E-mail: arnold@mathematik.uni-rostock.d400.de

†German Aerospace Research Establishment, Department of Vehicle System Dynamics, DLR Oberpfaffenhofen, Postfach 1116, D – 82230 Weßling, Germany, E-mail: netter@mkd8.df.op.dlr.de

the contact condition $g(q,t)=0$. The Lagrangian multipliers λ are determined by the second derivatives of the constraints:

$$\begin{aligned} 0 &= \frac{d^2}{dt^2} g(q,t) = \frac{d}{dt}(G(q,t)\dot{q} + g_t) \ , \\ &= [GM^{-1}](q)(f(q,\dot{q},\lambda,t) - G^T(q,t)\lambda) + g_{qq}(\dot{q},\dot{q}) + g_{qt}\dot{q} + g_{tt} \ . \end{aligned} \tag{2}$$

I. e., in general the contact conditions $g=0$ have to be (at least) two times continuously differentiable to get continuous constraint forces $-G^T\lambda$.

The shape of the contact patch due to the elastic deformation of wheel and rail is of great influence on the friction forces (i. e. on $f(q,\dot{q},\lambda,t)$). However, in the geometrical contact condition $g=0$ the elastic deformation (that is in the range of $0.1\,\mathrm{mm}$) can be neglected. This *rigid body contact model* avoids high-frequency oscillations in the solution and saves up to 95% cpu-time compared with a pure elastic contact model. Rigid body contact models have been used successfully in simulations with conic wheels (e. g. [14]) but they cause problems if the wheels have a *wear profile*. For these wheels (that are commonly used by European railway companies) the rigid body contact model results in constraints $g(q,t)=0$ that are either not differentiable or hard to evaluate (see Section 2).

In the dynamical simulation we therefore use a sufficiently differentiable approximation of $g(q,t)$ that can be evaluated efficiently. This approximation is implemented in a developer version of the simulation package SIMPACK, as part of SIMPACK it is used in the industrial design of railway vehicles ([12]). We carefully selected an approximation that is adapted to the contact problem between a single wheel and a rail, that is flexible w. r. t. future model changes and that can be computed and evaluated efficiently and very robustly (see Section 2).

The essential part of this approximation are polynomial 2D-tensorproduct splines minimizing a functional that combines a weighted least squares approximation of the rigid body contact model with a smoothing term. In Section 3 efficient algorithms for computation and evaluation of the tensorproduct splines are presented. Modern software environments allow a drastic acceleration of the computation process. In Section 4 we report on a parallel implementation on a cluster of workstations using PVM ([5]).

In the simulation package the tensorproduct splines are computed once in a pre-processing step, the spline coefficients are stored and allow the evaluation of the spline during the dynamical simulation. With 2 examples we illustrate in Section 5 the advantages of approximation in the dynamical simulation. Compared with the rigid body contact model the cpu-time for the simulation is reduced by a factor 5 ... 12.

2 Approximation of rigid body models for wheel-rail contact

As basic unit of wheel-rail contact we use the contact of a single wheel and a rail. This allows the simulation both for wheel-rail systems with classical rigid wheelsets and for modern advanced railway vehicles without rigid coupling between the wheels of a wheelset. In this section we describe 2 rigid body contact models and motivate the use of tensorproduct spline approximations in the contact model.

Throughout the present paper we restrict ourselves to problems with rails that have non-varying profiles along the track. This includes motion on straight tracks and motion into or in curves but excludes e. g. the motion of a railway vehicle running through a switch. Under this restriction the rails can be approximated locally by a straight track. We define a system of cartesian coordinates such that the x-axis is parallel to the track and the z-axis is oriented downwards. As rigid body the wheel has 6 degrees of freedom, its position relative to the rail is determined by $(x_{\mathrm{W}}, y_{\mathrm{W}}, z_{\mathrm{W}}, \varphi, \vartheta, \psi)$. According to our assumptions the cross-sections of the rail that are parallel to the y-z-plane do not vary along the track. Hence, the geometrical contact condition can be formulated independently of x_{W} (and independently of ϑ since the wheel is symmetric), it takes the form

$$z_{\mathrm{W}} = z_{\mathrm{W}}(y_{\mathrm{W}}, \varphi, \psi) \ . \tag{3}$$

The vertical displacement z_{W} of the wheel is determined by its lateral displacement y_{W} and by the angles φ, ψ that represent rotations around the x-axis and the z-axis, respectively.

The most straightforward contact model neglects the elastic deformation totally and treats wheel and rail as undeformed geometrical bodies ([14]). Then (3) says that the wheel lies on the rail and that the minimum distance between wheel and rail vanishes. In the special case $\psi = 0$ this minimum can be reached only for points P_{W} on the wheel surface that are directly below the symmetric axis of the wheel, i. e. below the centre of the wheel axle. This observation can be generalized ([10]): for arbitrary ψ the distance between wheel and rail can reach a minimum only in those points of the wheel surface where the normal vector to the surface is in the y-z-plane. This condition defines a curve

$$\mathfrak{C} = \mathfrak{C}(\psi) = \{ P_{\mathrm{W}}(s;\psi) \, : \, \underline{s} \le s \le \overline{s} \} \tag{4}$$

on the wheel surface that can be parametrized in general by the wheel coordinate s (the s-axis is parallel to the wheel axle). Along $\mathfrak{C}$ the *distance function* is defined by the distance of P_{W} to the rail surface:

$$\Delta(s; y_{\mathrm{W}}, \varphi, \psi) = e_3^T \cdot (P_{\mathrm{W}}(s) - P_{\mathrm{r}}(P_{\mathrm{W}}(s))) \tag{5}$$

(the distance is measured along the z-axis that is orientated downwards). Here $P_{\mathrm{r}}(P_{\mathrm{W}})$ denotes a point on the rail surface with the same x- and y-coordinates as P_{W} (note, that Δ is independent of z_{W}). The undeformed geometrical bodies of wheel and rail are in contact iff

$$z_{\mathrm{W}} + \max_s \Delta(s; y_{\mathrm{W}}, \varphi, \psi) = 0 \ . \tag{6}$$

There are several ways to evaluate this contact condition efficiently ([14, 10]) but its main drawback is obvious: (6) defines a continuous constraint that is in general only piecewise differentiable. A point $P_{\mathrm{W}}(s^*)$ with $s^* = s^*(y_{\mathrm{W}}, \varphi, \psi)$ and $\Delta(s^*; y_{\mathrm{W}}, \varphi, \psi) = \max_s \Delta(s; y_{\mathrm{W}}, \varphi, \psi)$ is called a *common point* (of wheel and rail). A typical phenomenon in the contact of rails and wheels with wear profiles are *jumps* of the common point, i. e. s^* does not vary continuously with $(y_{\mathrm{W}}, \varphi, \psi)$ ([10], see also Fig. 1). If s^* changes discontinuously then (6) is in general not differentiable. Eqs. (1) with contact conditions (6) do not have a (classical) solution.

In [1] several attempts have been made to avoid these difficulties. Instead of neglecting the elastic deformation totally the modified contact model neglects only the quantitative effects but considers the elastic deformation qualitatively. We replace (6) by

$$z_{\mathrm{W}} + \mathrm{smax}_s^{(\nu)} \Delta(s; y_{\mathrm{W}}, \varphi, \psi) = 0 \ . \tag{7}$$

with

$$\mathrm{smax}_s^{(\nu)} \Delta(s; y_{\mathrm{W}}, \varphi, \psi) := \nu \ln \Big(\frac{1}{\overline{s} - \underline{s}} \sum_{i=1}^{N} (s_i - s_{i-1}) \exp(\frac{1}{\nu} \Delta(s_i; y_{\mathrm{W}}, \varphi, \psi)) \Big) \ .$$

Here $\nu > 0$ denotes a small parameter and $\{s_i\}$ with $\underline{s} = s_0 < s_1 < \ldots < s_N = \overline{s}$ a grid on $[\underline{s}, \overline{s}]$. For $\nu \to 0$ function $\mathrm{smax}^{(\nu)}$ converges pointwise to $\max_s \Delta(s; y_{\mathrm{W}}, \varphi, \psi)$. If $\nu > 0$ then the contact condition (7) is as differentiable as the distance function Δ, i. e. we get sufficient regularity in the model equations (1). The price for this regularity is high: one evaluation of $\mathrm{smax}^{(\nu)}$ requires N evaluations of the distance function. Depending on ν this number N has to be (very) large to get a reasonable contact condition, typical values are $10^3 \ldots 10^4$.

In the following we refer to (6) and (7) as the *basic* and the *regularized* (contact) *model*, respectively. Besides the geometrical contact condition we have to define the point on the wheel surface in that constraint force and friction force operate. For historical reasons this point is called *contact point* $P_{\mathrm{W}}(S)$ with $S = S(y_{\mathrm{W}}, \varphi, \psi)$. In the basic model it coincides with the common point (take the one with smallest s if there are more than one common points). In the regularized model we choose S as weighted mean value of s along $\mathfrak{C}$ with weights $\exp(\frac{1}{\nu}\Delta(s; y_{\mathrm{W}}, \varphi, \psi))$ (see Fig. 2).

To avoid the drawbacks of both rigid contact models we use in the dynamical simulation a weighted least squares approximation of (6) or of (7). The approximation error is kept in the range of the model error (that is $\ll 1\,\mathrm{mm}$ in the geometrical contact condition). One key observation is that in comparison with y_{W} and φ the influence of the yaw angle ψ on the distance funtion Δ is very small. Furthermore, $|\psi|$ is small during simulation and Δ is an odd function w. r. t. ψ. These facts allow to decouple the 3D problem "approximate $f(y_{\mathrm{W}}, \varphi, \psi)$ " into several 2D approximation problems with fixed yaw angles ψ. (Here $f(y_{\mathrm{W}}, \varphi, \psi) := \max_s \Delta(s; y_{\mathrm{W}}, \varphi, \psi)$ in the basic model and $f(y_{\mathrm{W}}, \varphi, \psi) := \mathrm{smax}_s^{(\nu)} \Delta(s; y_{\mathrm{W}}, \varphi, \psi)$ in the regularized model.) Currently we use

$$f(y_{\mathrm{W}}, \varphi, \psi) \approx (1 - (\psi/\hat{\psi})^2) f(y_{\mathrm{W}}, \varphi, 0) + (\psi/\hat{\psi})^2 f(y_{\mathrm{W}}, \varphi, \hat{\psi})$$

with $\hat{\psi} := 3^\circ$ and approximate $f(y_{\mathrm{W}}, \varphi, 0)$ and $f(y_{\mathrm{W}}, \varphi, \hat{\psi})$ separately.

From the great variety of possible approximations for a function $f(y_{\mathrm{W}}, \varphi, \hat{\psi})$ with fixed $\hat{\psi}$ that still depends on the 2 variables y_{W} and φ we selected a polynomial spline ([3]) since the approximating spline can be computed solving a system of *linear* equations and it does not need any a priori information about the function to be approximated. This approximation is very robust, a potential user does not need any special knowledge about the algorithm and the algorithm itself can be used for many different sets of input data, i. e. it is open to future changes in the model. On the other hand the spline approximation is flexible: approximation error and differentiability of the spline can be controlled using splines with appropriate order and grid.

The regular pattern of the data in Fig. 2 motivates the restriction to tensorproduct splines on rectangular grids to simplify the computation of the spline. I. e. along each gridline in a $(y_{\mathrm{W}}, \varphi)$-grid the approximation coincides with a standard 1D spline function. It is well known that discontinuities in the data need special care in the approximation process. We therefore fit the tensorproduct spline $\sigma(y_{\mathrm{W}}, \varphi)$ not only to the given function f but add a smoothing term. More precisely, the spline σ is defined such that it minimizes a discrete version of the functional

$$\iint_\Omega \omega(y_{\mathrm{W}}, \varphi)\Big(\sigma(y_{\mathrm{W}}, \varphi) - f(y_{\mathrm{W}}, \varphi, \hat{\psi})\Big)^2 dy_{\mathrm{W}} d\varphi + \iint_\Omega (\mu_1^2 \frac{\partial^2 \sigma}{\partial y_{\mathrm{W}}^2} + 2\mu_1\mu_2 \frac{\partial^2 \sigma}{\partial y_{\mathrm{W}} \partial \varphi} + \mu_2^2 \frac{\partial^2 \sigma}{\partial \varphi^2})\, dy_{\mathrm{W}} d\varphi \quad (8)$$

with fixed non-negative parameters $\mu_1, \mu_2 \in \mathbb{R}$ and a (positive) *weight function* $\omega(y_{\mathrm{W}}, \varphi)$ that allows to introduce a priori knowledge about the given function f (see the next section for details). The smoothing term is a generalization of the Thin–Plate functional.

Example 1 In a neighbourhood of the jumps of the contact point in Fig. 1 it does not make sense to compute a polynomial spline that is close to the (discontinuous) data, i. e. $\omega(y_{\mathrm{W}}, \varphi)$ should be small in these regions. Fig. 3 shows a typical weight function for these data.

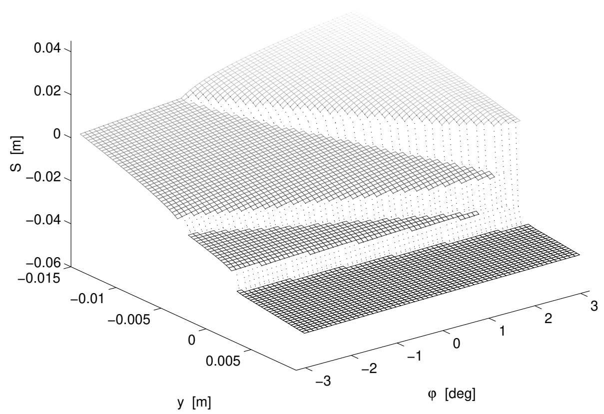

Figure 1: Basic contact model: contact point $S(y_{\mathrm{W}}, \varphi, 1.0°)$.

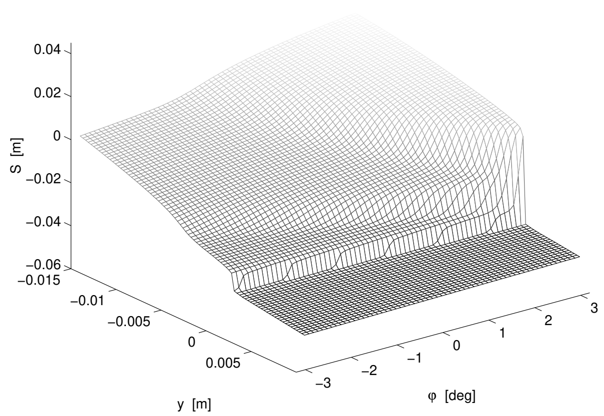

Figure 2: Regularized contact model: contact point $S(y_{\mathrm{W}}, \varphi, 1.0°)$, $\nu = 2.0_{10} - 5$.

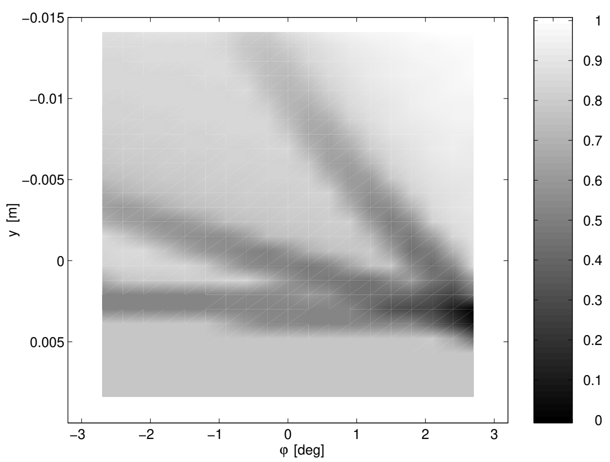

Figure 3: Typical weight function $\omega(y_{\mathrm{W}}, \varphi)$ in least squares approximation.

3 The computation of the approximating 2D-tensorproduct spline

Computing the 2D-tensorproduct spline we have to determine a few hundred spline coefficients on the basis of a few thousand data points $f_l = f(y_{\mathrm{W},l}, \varphi_l, \hat{\psi})$. Using a B-spline representation the spline coefficients are obtained as least squares solution of a large sparse overdetermined system of linear equations. In this section we form the overdetermined system of linear equations and describe an algorithm for the computation of the least squares solution that exploits the sparsity structure. In some examples we compare the spline approximation with the original data f_l and illustrate the use of the smoothing term.

To describe the algorithm we write the spline σ as linear combination of B-splines ([3]) adopting ideas and the notation of [13]. Let $k \in \mathbb{N}$ and knot sequences $t^{(i)}$, $(i = 1, 2)$ be given with $k \geq 2$ and

$$t_1^{(i)} = \ldots = t_k^{(i)} = \underline{x}^{(i)} < t_{k+1}^{(i)} \leq \ldots \leq t_{n_i}^{(i)} < \overline{x}^{(i)} = t_{n_i+1}^{(i)} = \ldots = t_{n_i+k}^{(i)} .$$

The sequences $\{ B_{kj}^{(i)} \}$ of normalized B-splines of order k with respect to the knot sequences $t^{(i)}$ are recursively defined by

$$\begin{aligned} B_{1j}^{(i)}(x) &\equiv 0 \quad \text{if } j < k \text{ or } j > n_i , \\ B_{1j}^{(i)}(x) &= \begin{cases} 1 & \text{for } x \in I_j , \\ 0 & \text{otherwise,} \end{cases} \qquad (j = k, \ldots, n_i) \end{aligned}$$

with $I_j := [t_j^{(i)}, t_{j+1}^{(i)})$ for $j = k, \ldots, n_i - 1$, $I_{n_i} := [t_{n_i}^{(i)}, t_{n_i+1}^{(i)}]$ and

$$B_{kj}^{(i)} := \omega_{kj}^{(i)} B_{k-1,j}^{(i)} + (1 - \omega_{kj}^{(i)}) B_{k-1,j+1}^{(i)} \tag{9}$$

where

$$\omega_{kj}^{(i)}(x) := \begin{cases} \frac{x - t_j^{(i)}}{t_{j+k-1}^{(i)} - t_j^{(i)}} & \text{if } t_{j+k-1}^{(i)} - t_j^{(i)} > 0 , \\ 0 & \text{otherwise.} \end{cases}$$

The approximating spline is given by

$$\sigma(x^{(1)}, x^{(2)}) = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} B_{k,j_1}^{(1)}(x^{(1)}) B_{k,j_2}^{(2)}(x^{(2)}) \alpha_{j_1 j_2} \tag{10}$$

with spline coefficients $\alpha_{j_1 j_2}$.

Remark 1 a) The tensorproduct splines $\{ B_{k,j_1}^{(1)} B_{k,j_2}^{(2)} \; : \; 1 \leq j_i \leq n_i, \; i = 1, 2 \}$ are a basis of the $(n_1 \cdot n_2)$-dimensional vectorspace of all spline functions σ defined by (10), they satisfy

$$\sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} B_{k,j_1}^{(1)}(x^{(1)}) B_{k,j_2}^{(2)}(x^{(2)}) \equiv 1 , \quad (x^{(1)} \in [\underline{x}^{(1)}, \overline{x}^{(1)}], \; x^{(2)} \in [\underline{x}^{(2)}, \overline{x}^{(2)}])$$

and have final support

$$\text{supp } (B_{k,j_1}^{(1)}(x^{(1)}) B_{k,j_2}^{(2)}(x^{(2)})) = [t_{j_1}^{(1)}, t_{j_1+k}^{(1)}] \times [t_{j_2}^{(2)}, t_{j_2+k}^{(2)}] . \tag{11}$$

b) We restrict ourselves to grids with $t_k^{(i)} < t_{k+1}^{(i)} < \ldots < t_{n_i+1}^{(i)}$ to get a $(k-2)$-times continuously differentiable spline function. Typical orders are $k = 5$ for the approximation of the contact condition and $k = 3$ for the approximation of the contact point S.

c) To simplify notation we set $x^{(1)} := y_{\mathrm{W}}$, $x^{(2)} := \varphi$ in this section. In the actual computations a slightly improved version is used, see Remark 3b.

The first term in (8) is the weighted L_2-norm $\|\sigma - f\|_{2,\omega}^2$. This term is substituted by its discrete analogue

$$\Phi_1(\sigma) := \sum_{l=1}^{m} \omega_l \Big(\sigma(x_l^{(1)}, x_l^{(2)}) - f_l \Big)^2 \tag{12}$$

with given points $(x_l^{(1)}, x_l^{(2)})$, $(l = 1, \ldots, m)$ and function values $f_l = f(x_l^{(1)}, x_l^{(2)}) = f(y_{\mathrm{W},l}, \varphi_l, \hat{\psi})$ that are either evaluated during the computation of σ or read from a file. In general there are much more data points than free spline coefficients: $m \gg n_1 n_2$. To consider the distribution of data points in the $x^{(1)}$-$x^{(2)}$-plane we define for each rectangle

$$R_{j_1 j_2} = [t_{j_1+k-1}^{(1)}, t_{j_1+k}^{(1)}) \times [t_{j_2+k-1}^{(2)}, t_{j_2+k}^{(2)})$$

the number $N_{j_1 j_2}$ of data points in $R_{j_1 j_2}$ and the area $A_{j_1 j_2}$ of $R_{j_1 j_2}$. If $(x_l^{(1)}, x_l^{(2)}) \in R_{j_1 j_2}$ then we use the weight

$$\omega_l = \frac{1}{N_{j_1 j_2}} \omega(y_{\mathrm{W},l}, \varphi_l) \cdot A_{j_1 j_2} \ .$$

Inserting (10) function $\Phi_1(\sigma)$ in (12) can be written as

$$\Phi_1(\sigma) = \sum\nolimits_l \omega_l \Big(\sum\nolimits_{j_1, j_2} B_{k,j_1}^{(1)}(x_l^{(1)}) B_{k,j_2}^{(2)}(x_l^{(2)}) \alpha_{j_1 j_2} - f_l \Big)^2 = \|B\alpha - d\|_2^2$$

with the unknown spline coefficients

$$\alpha = (\, \alpha_{11}, \alpha_{12}, \ldots, \alpha_{1n_2}, \alpha_{21}, \ldots, \alpha_{n_1 n_2} \,)^T \in \mathbb{R}^{n_1 n_2} \ , \tag{13}$$

a matrix $B \in \mathbb{R}^{m \times (n_1 n_2)}$ and a vector $d \in \mathbb{R}^m$, that contains the (weighted) values f_l. Each data point $(x_l^{(1)}, x_l^{(2)})$ defines one row of B and d. Because of (11) at most k^2 terms in $\sigma(x^{(1)}, x^{(2)}) = \sum_{j_1, j_2} B_{k,j_1}^{(1)}(x^{(1)}) B_{k,j_2}^{(2)}(x^{(2)}) \alpha_{j_1 j_2}$ do not vanish. I. e. each row of matrix B has (at most) k^2 non-zero elements, all rows corresponding to data points in one and the same rectangle $R_{j_1 j_2}$ have the same sparsity pattern. This gives matrix B a very regular sparsity structure that is illustrated by the left plot of Fig. 4. This academic example was chosen such that it still fits into a MatLab plot. The matrix has 6 large blocks (corresponding to the $n_1 - k + 1 = 6$ rectangles in $x^{(1)}$-direction), each of them contains 4 smaller blocks (corresponding to the $n_2 - k + 1 = 4$ rectangles in $x^{(2)}$-direction).

To solve $\|B\alpha - d\|_2^2 \to \min$ for α we transform B to $\binom{\hat{B}}{0}$ with an upper triangular matrix $\hat{B} \in \mathbb{R}^{(n_1 n_2) \times (n_1 n_2)}$ using orthogonal transformations. Here the sparsity of B is exploited in two ways: At first all data points in one rectangle $R_{j_1 j_2}$ give a (dense) scheme of $N_{j_1 j_2} \times k^2$ non-zero matrix elements. Applying (fast) Givens rotations to these k^2 columns this scheme is transformed to upper triangular form (see the central plot of Fig. 4 for an example with $k = 4$ after processing the data points in R_{11} and R_{12}). We refer to this transformation as Algorithm I.

At second the transformed blocks of $\leq k^2$ rows with (at most) k^2 non-zero elements per row are combined rectangle by rectangle (Algorithm II). We start with the output of Algorithm I for rectangle R_{11} that defines a matrix $\binom{\hat{B}_{11}}{0} \in \mathbb{R}^{m \times (n_1 n_2)}$ with an upper triangular matrix $\hat{B}_{11} \in \mathbb{R}^{(n_1 n_2) \times (n_1 n_2)}$ that has k^2 non-zero columns and (at most) k^2 non-zero rows. The output of Algorithm I for rectangle R_{12} is appended to this matrix. Applying (fast) Givens rotations this combined matrix is transformed to a matrix $\binom{\hat{B}_{12}}{0} \in \mathbb{R}^{m \times (n_1 n_2)}$ with upper triangular matrix

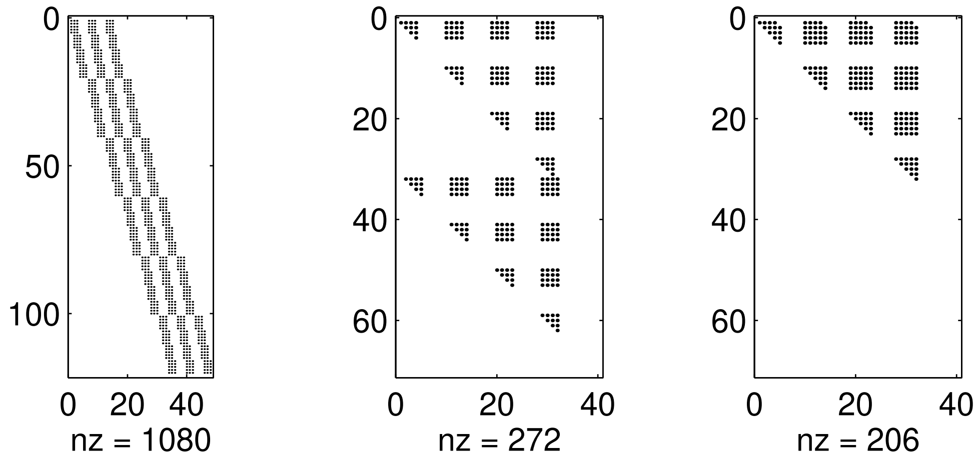

Figure 4: Exploiting sparsity in the solution of the overdetermined system (17). left: structure of B if $k=3$, $n_1=8$, $n_2=6$, $m=5(n_1-k+1)(n_2-k+1)=120$; centre: ($k=4$, $n_2=9$) upper left part of the matrix at the moment when Algorithm I has been applied to R_{11} and R_{12}; right: ($k=4$, $n_2=9$) upper left part of $\hat{B}_{12}$.

$\hat{B}_{12} \in \mathbb{R}^{(n_1 n_2)\times(n_1 n_2)}$ (see central and right plot of Fig. 4). We proceed with R_{13} and so on until all rectangles have been processed.

The resulting transformed matrix $\hat{B}$ is banded with bandwidth $n_2(k-1)+k$. Obviously in Algorithm II only the non-zero rows have to be kept in memory, i. e. storage has to be provided not for the (very large) matrix B but for $\hat{B}$, only. In the same way Algorithm I works with storage for a $k^2\times k^2$-matrix and a row vector of length k^2 that contains the elements for the next data point. The main difference between both algorithms is the fact that Algorithm I works with vectors of length $\le k^2$ and Algorithm II processes vectors with up to $n_2(k-1)+k$ non-zero elements.

Now we consider the smoothing term in (8). Using (13) and the Kronecker product notation definition (10) can be rewritten as

$$\sigma(x^{(1)},x^{(2)}) = (\, B_k^{(1)}(x^{(1)}) \otimes B_k^{(2)}(x^{(2)})\,)\cdot\alpha$$

with

$$B_\kappa^{(i)}(x^{(i)}) = (\, B_{\kappa,k+1-\kappa}^{(i)}(x^{(i)}),\ldots,B_{\kappa,n_i}^{(i)}(x^{(i)})\,)\ ,\quad (i=1,2;\ \kappa=1,\ldots,k)\ .$$

The derivatives of σ can be expressed by B-splines of lower order (see e. g. [13]):

$$\frac{\partial^{r_1+r_2}\sigma}{\partial(x^{(1)})^{r_1}\,\partial(x^{(2)})^{r_2}} = \left(B_{k-r_1}^{(1)}(x^{(1)})\otimes B_{k-r_2}^{(2)}(x^{(2)})\right)\cdot D_{r_1r_2}\alpha\ .$$

Here $D_{r_1r_2} := D_{r_1}^{(1)}\otimes D_{r_2}^{(2)}$ and $D_r^{(i)}\in\mathbb{R}^{(n_i-r)\times n_i}$ is defined by

$$D_0^{(i)} := I\ ,\quad D_r^{(i)} := H_r^{(i)}L_r^{(i)}H_{r-1}^{(i)}L_{r-1}^{(i)}\cdots H_1^{(i)}L_1^{(i)}$$

with

$$H_\nu^{(i)} := (k-\nu)\ \mathrm{diag}\left(\frac{1}{t_{j+k-\nu}^{(i)}-t_j^{(i)}}\right)_{1+\nu\le j\le n_i}\in\mathbb{R}^{(n_i-\nu)\times(n_i-\nu)}$$

and

$$L_\nu^{(i)} := \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{pmatrix} \in \mathbb{R}^{(n_i-\nu)\times(n_i-\nu+1)} .$$

The L_2-norm of the partial derivative can be computed from

$$\iint_\Omega \Big(\frac{\partial^{r_1+r_2}\sigma}{\partial(x^{(1)})^{r_1}\,\partial(x^{(2)})^{r_2}} \Big)^2 dx^{(1)}dx^{(2)} = \alpha^T D_{r_1r_2}^T M_{r_1r_2} D_{r_1r_2}\alpha \tag{14}$$

with the Gram matrix

$$M_{r_1r_2} = \iint_\Omega \Big(B_{k-r_1}^{(1)}(x^{(1)}) \otimes B_{k-r_2}^{(2)}(x^{(2)}) \Big)^T \Big(B_{k-r_1}^{(1)}(x^{(1)}) \otimes B_{k-r_2}^{(2)}(x^{(2)}) \Big)\, dx^{(1)}dx^{(2)} . \tag{15}$$

Matrix $M_{r_1r_2}$ is symmetric and positive definite. The computation of the integrals in (15) is avoided if we use as smoothing term instead of the L_2-norm a discrete analogue substituting $M_{r_1r_2}$ in (14) by $F_{r_1r_2}^T F_{r_1r_2}$ with $F_{r_1r_2} := F_{r_1}^{(1)} \otimes F_{r_2}^{(2)}$ and

$$F_r^{(i)} := (H_r^{(i)})^{-1/2} = \mathrm{diag}\Big(\sqrt{\frac{t_{j+k-r}^{(i)} - t_j^{(i)}}{k-r}} \Big)_{r+1\le j\le n_i} \in \mathbb{R}^{(n_i-r)\times(n_i-r)} .$$

This is the 2D version of an idea that is used in 1D e. g. in [13].

Summarizing all these notations the smoothing term in (8) is substituted by

$$\Phi_2(\sigma) := \mu_1^2\|F_{20}D_{20}\alpha\|^2 + 2\mu_1\mu_2\|F_{11}D_{11}\alpha\|^2 + \mu_2^2\|F_{02}D_{02}\alpha\|^2 \tag{16}$$

and α is computed such that $\Phi_1(\sigma) + \Phi_2(\sigma) \to \min$. I. e. α is least squares solution of the overdetermined system

$$\begin{pmatrix} B \\ \mu_1 F_{20} D_{20} \\ \sqrt{2\mu_1\mu_2} F_{11} D_{11} \\ \mu_2 F_{02} D_{02} \end{pmatrix} \alpha = \begin{pmatrix} d \\ 0 \\ 0 \\ 0 \end{pmatrix} . \tag{17}$$

If the smoothing term does not vanish and $\mu_1 \neq 0$, $\mu_2 \neq 0$ then the unique solvability of the least squares problem $\Phi_1(\sigma) + \Phi_2(\sigma) \to \min$ is guaranteed under mild assumptions, e. g. if there is at least one data point per rectangle $R_{j_1j_2}$. The smoothing term $\Phi_2(\sigma)$ introduces $3n_1n_2 - 3(n_1+n_2) + 1$ additional equations in (17). The sparsity structure of matrices $D_{r_1r_2}$ allows to assign each of these equations in a natural way to one of the rectangles $R_{j_1j_2}$, i. e. the additional equations can be processed by a modification of Algorithm I (with 3 additional equations in most of the rectangles). We skip the details of this modification since it is more or less straightforward but needs a lot of additional notations.

In this way (17) is transformed to a system with upper triangular matrix of coefficients, the solution α is obtained by a back-substitution ([6, Chapter 5]). The application of orthogonal transformations and the B-spline representation make the numerical solution robust ([6], [3]). The algorithm is not sensitive w. r. t. failures in the evaluation of function values f_l in (12). If for any reason function f can not be evaluated at some $(x_1^{(l)}, x_2^{(l)})$ then the corresponding row in B and d is simply deleted. Once the spline coefficients α are computed the spline σ can be evaluated efficiently and in a numerically stable way from (9) and (10).

The efficiency of the proposed Algorithms I and II is discussed in Section 4. We end this part with some examples. In the tests we use the same coordinates $(x^{(1)}, x^{(2)})$, the same type of grids and the same weight function as in the Pre-Processor of the simulation software (see Remark 3 for more details). To compare spline and given data we prefer 2D plots since they show more details. In the plots of Figs. 5–7 the roll angle φ is fixed to $\varphi = 1.43°$ corresponding to the nominal position of a rigid wheelset on a straight track. The splines are computed on a grid with 20×10 rectangles, the grid in $x^{(2)}$-direction is equidistant, the grid in $x^{(1)}$-direction is based on an equidistant grid with 15 intervals (see Remark 3).

Fig. 5 shows for $\psi = 0$ the vertical displacement computed from the basic and the regularized contact model, and the spline approximating the basic contact model (for the regularized contact model given data and spline coincide in the plot accuracy). The jumps of the common point, i. e. the discontinuities of the basic model, are marked by circles. The 2D-tensorproduct spline with $\mu_1 = \mu_2 = 10^{-8}$ and $k = 5$ is three times continuously differentiable, the approximation error remains in the size of the model error of the rigid contact models.

Because of discontinuities in the basic model and derivatives of large absolute value in the regularized model the smoothing term is extremely important. This is nicely illustrated by Figs. 6 and 7 that show for $\psi = 1.0°$ the contact point and its spline approximation with order $k = 3$ once for the basic and once for the regularized contact model. Without the smoothing term the approximation oscillates and can not be used in the dynamical simulation. Adding the smoothing term with $\mu_1 = \mu_2 = 10^{-6}$ a rather good approximation of the contact point is obtained. These results illustrate that the 2D-tensorproduct spline gives really useful approximations for the geometrical data of wheel-rail contact.

4 Exploiting parallelism in the computation of the spline coefficients

After the tensorproduct spline has been computed it can be used in the dynamical simulation again and again if wheel and rail profile do not change. So it is not too important to get the spline coefficients extremely fast, cpu-times in the range of minutes are acceptable. Nevertheless the numerical effort is large if in (12) the evaluation of f_l is very time-consuming (e. g. large N in (7)) and / or the number m of data points is large. To keep cpu-time small we implemented a parallel version of the proposed algorithm on a cluster of workstations. In this section we describe the parallelization in detail and report on the performance on a homogeneous and on a heterogeneous network.

Remark 2 Computing the approximating spline the overall cpu-time covers both the costs for m function evaluations f_l in (12) and the costs for the solution of the overdetermined system (17). If there are e. g. in each rectangle $N_{j_1 j_2} = s \cdot k - 3$ data points (i. e. $m = (sk-3)(n_1 - k + 1) \cdot (n_2 - k + 1)$) then Algorithms I and II solve (17) with

$$\begin{cases} \mathcal{O}\Big(C_{\text{Giv}} \frac{m}{2} (k^4 + (n_2^2 - k^2)(k^2 - s(k - \frac{s}{3})))\Big) & \text{operations if } s \le k \, , \\ \mathcal{O}\Big(C_{\text{Giv}} (\frac{m}{2} k^4 + (n_2^2 - k^2) \cdot \frac{1}{6} k^4 (n_1 - k + 1)(n_2 - k + 1))\Big) & \text{operations if } s \ge k \, . \end{cases}$$

Here $C_{\text{Giv}} \cdot r$ denotes the effort for the (fast) Givens transformation of two r-dimensional vectors, $C_{\text{Giv}} = 6$ for classical and $C_{\text{Giv}} = 4$ for fast Givens rotations ([6, Chapter 5.1]). Discussing the numerical effort we can restrict ourselves to meshs of moderate size (typical values are $n_1 = 15 \ldots 30$, $n_2 = 10 \ldots 20$). Extremely fine meshs would decrease the efficiency

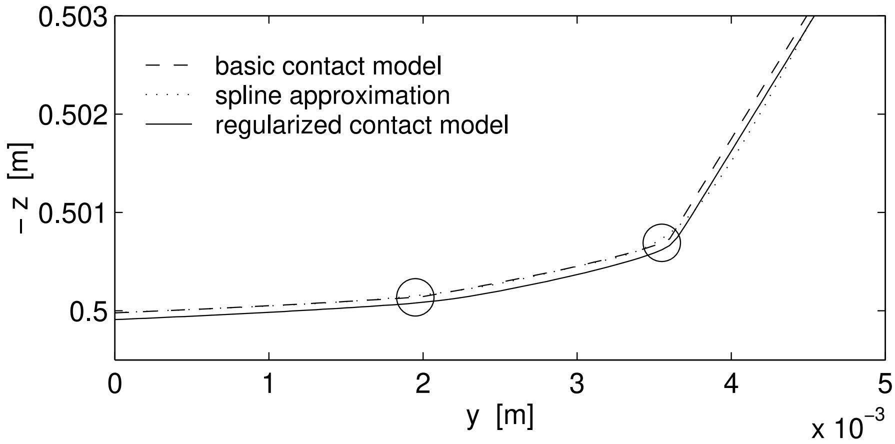

Figure 5: Contact model: basic model, regularized model, and spline approximation.

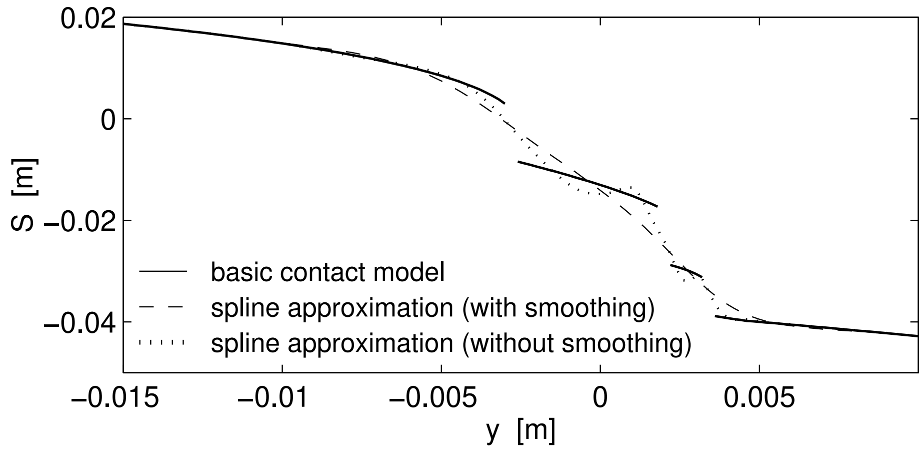

Figure 6: Basic contact model ($\varphi = 1.43°$, $\psi = 1.0°$): approximation of contact point S.

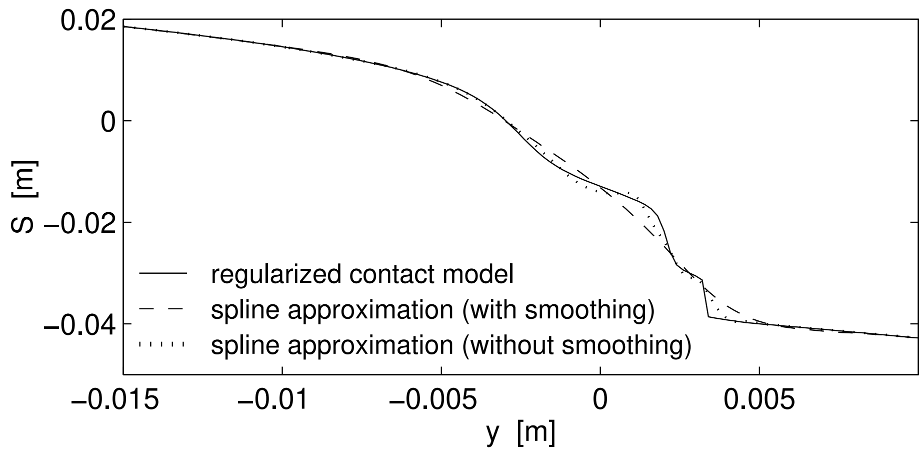

Figure 7: Regularized contact model ($\varphi = 1.43°$, $\psi = 1.0°$): approximation of contact point S.

of the simulation software since large arrays of spline coefficients would have to be handled in the simulation package. Looking at the real physical system there is no need for such fine meshs since they could reduce only the approximation error in (8) but not the model error of the rigid contact model. Accelerating the computation we therefore focus on the situation that the function evaluations f_l are very time-consuming. The above estimates illustrate that from the point of view of cpu-time this is equivalent to the situation that m is extremely large.

The linear algebra part of the costs could be reduced drastically if we would restrict ourselves to $\omega_l = \omega^{(1)}(x^{(1)}) \cdot \omega^{(2)}(x^{(2)})$ and to data points $(x_1^{(l)}, x_2^{(l)})$ on a rectangular mesh in (12) since matrix B in (17) has in this special case a tensorproduct structure similar to $F_{r_1 r_2}$, $D_{r_1 r_2}$ that can be exploited in the solution process (see e. g. [4, Chapter 10]). In our application the restriction to special weight functions seems to be inacceptable (see Fig. 3). Furthermore the savings in the *overall* costs are small if the costs for the evaluation of f_l dominate. For both reasons we do not follow this mesh data approach.

Instead of reducing the numerical effort for the linear algebra part we distribute the whole work among a few workstations that are coupled to a cluster via Internet. The algorithm of Section 3 consists of the sequential part (Algorithm II) and of Algorithm I that can be applied in parallel to the $(n_1 - k + 1)(n_2 - k + 1)$ rectangles of the mesh. This motivates a master–slave model of parallelization. In a cluster of n_{host} workstations one master does the sequential work and sends jobs to $n_{\text{host}} - 1$ slaves. All these "jobs" have identical structure: for a given rectangle $R_{j_1 j_2}$ the slave evaluates the function values f_l and applies Algorithm I to get the upper triangular scheme of coefficients that is sent back to the master.

It is hard to predict the actual performance of a slave process in a multi-user workstation environment. That is why we use a dynamic load balance scheme, i. e. fast slaves get more jobs than slow ones ([5, Section 8.3]). A slave is *idle* before it gets the first job and after it has done one job and not yet received the next one. In a loop the master

- checks which slaves are idle,
- sends jobs to all idle slaves,
- waits for results from a slave (i. e. for the result of Algorithm I applied to one rectangle $R_{j_1 j_2}$), and
- receives these results and applies Algorithm II

until all rectangles of the mesh have been processed. Finally, in a last sequential step the banded upper triangular system $\hat{B}\alpha = \hat{d}$ is solved by back-substitution ($\hat{B} \in \mathbb{R}^{(n_1 n_2) \times (n_1 n_2)}$, $\hat{d} \in \mathbb{R}^{n_1 n_2}$). To handle the (rare) situation that one slave has got a job but gives no response the master continues sending jobs until Algorithm I has been successfully applied to *all* rectangles. So we avoid that the overall computation could be blocked by one blocked slave, the additional costs are negligible (Algorithm I might be applied twice — but in parallel — to a few rectangles).

Furthermore the sequential Algorithm II is adapted to the dynamic load balance scheme. In the basic form of Section 3 the rectangles are processed in increasing order R_{11}, R_{12}, $\ldots$, R_{1,n_2-k+1}, R_{21}, $\ldots$. In the parallel version the jobs are sent in this order but we can not expect to get the results from the slaves in the same order. Let us consider the moment when the master has processed the results for rectangle $R_{j_1 j_2}$ by Algorithm II (suppose $j_2 < n_2 - k + 1$ for simplicity of presentation). In the sequential form of the algorithm the master would have to wait until the corresponding slave sends the result of Algorithm I applied to R_{j_1, j_2+1} — even if there are already available results for other rectangles than R_{j_1, j_2+1} from faster slaves.

Especially on heterogeneous networks the performance is improved if the master does not wait, skips rectangle R_{j_1,j_2+1} at the moment and continues with the next available rectangle. Later, as soon as the results for R_{j_1,j_2+1} have been sent from the slave, they are processed by the master in the usual way. This asynchronization has no influence on the final result of the computations but the sequential work increases slightly. The overall cpu-time is nevertheless reduced since the master is not blocked by a (very) slow slave (see Example 3).

This master–slave model was implemented using PVM ([5]). The sequential code was extended by some subroutines that handle the dynamic load balance scheme and the communication between master and slaves. The communication costs of the parallel algorithm are small, the master sends only scalars to the slaves $(j_1, j_2, \dots)$ and receives matrices of size $k^2 \times k^2$. To rate the performance of the parallel algorithm we compare the sequential program with two parallel versions that differ by the hardware in use. The sequential algorithm runs on a SUN Sparc5 that is also used as master in the parallel version. We couple up to three additional Sparc5 stations with identical hard- and software to this master to get the *homogeneous* network. At the weekend we have more resources available and include two HP workstations to the network (*heterogeneous* network). We expect one of them (hp 735/99) to be much faster than the Sparc5's, the other one is older (hp 710) and has a hard job as `ftp`–server, it represents a potential bottleneck in our network.

The parallelization was mainly motivated by time-consuming function evaluations f_l, in future models f_l could e. g. even contain solutions of integral equations. In the experiments a quantitative comparison of the algorithms is simplified if we fix the function f that is to be approximated, and increase m, i. e. the number of data points (see also Remark 2). In the following numerical tests the contact condition (7) with $N = 200$ is approximated on a mesh with 20×10 rectangles. We chose the number m of data points equal to the number of equations in (17), i. e. in this comparison the (small) additional effort for the smoothing term is not considered.

Example 2 Figs. 8 and 9 show for varying m the cpu-time of the sequential algorithm ("$\times$"), of the parallel algorithm on the homogeneous network with one ("$\circ$"), two ("$+$"), and three ("$*$") slaves, respectively, and of the parallel algorithm on the heterogeneous network with 5 slaves ("$\otimes$"). The figures differ by the order k of the spline: we have $k = 5$ (i. e. $n_1 = 24$, $n_2 = 14$) in Fig. 8 and $k = 3$ (i. e. $n_1 = 22$, $n_2 = 12$) in Fig. 9. The bandwidths of the resulting upper triangular systems are 61 and 27, respectively. In both cases the benefits of the parallelization are obvious for large m. For small m we do not expect a substantial speed-up by parallelization because the sequential part of the linear algebra costs dominates the overall costs. Table 1 corresponds to Fig. 8 and summarizes the speed-up of the parallel algorithm. Asymptotically for $m \to \infty$ a speed-up of $\approx n_{\text{host}} - 1$ is reached on the homogeneous network of n_{host} workstations (i. e. with $n_{\text{host}} - 1$ slaves).
Table 1 and Fig. 8 show nicely the advantages of a homogeneous network that equidistributes the jobs among the slaves (i. e. approximately 67 jobs per slave in the network with 3 slaves). The heterogeneous network results in different loads for the slaves (a typical load balance are 37, 38, and 39 jobs on the Sparc5's, 59 jobs on the hp 735/99, and 29 jobs on the hp 710). Compared with the homogeneous network without the HP's a substantial speed-up is obtained for $m \geq 3000$, only. For very large m, however, it is clearly superior (see also Fig. 9).
For smaller spline order parallelization reduces cpu-time already for small m. Compared with Fig. 8 we have in Fig. 9 less sequential work (because of the smaller bandwidth) and less communication costs ($k^2 = 9$ instead of $k^2 = 25$). For large m both figures are almost identical since linear algebra and communication costs are dominated by the m function evaluations f_l.

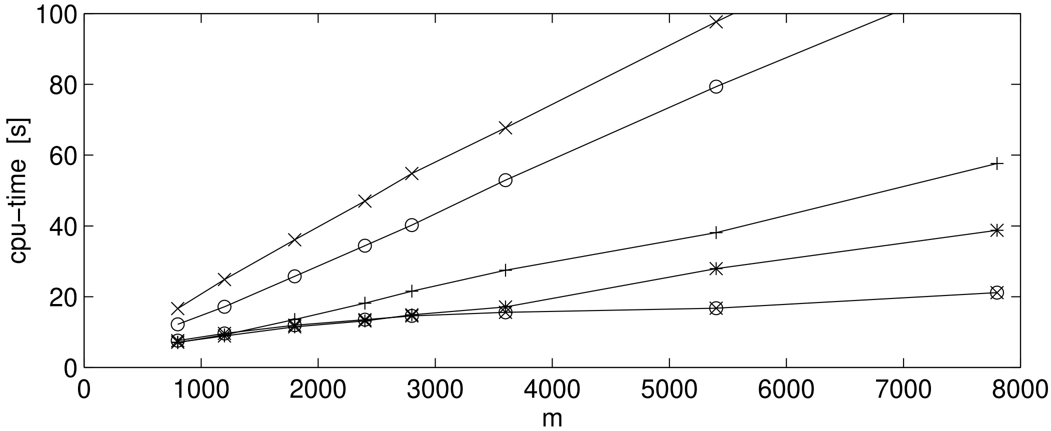

Figure 8: Numerical effort of sequential and parallel algorithms, $k = 5$, $n_1 = 24$, $n_2 = 14$.

m	800	1200	1800	2400	2800	3600	5400	7800
1 slave	1.37	1.45	1.40	1.37	1.36	1.28	1.23	1.21
2 slaves	2.37	2.71	2.66	2.59	2.54	2.46	2.56	2.35
3 slaves	2.34	2.79	3.14	3.58	3.68	3.96	3.50	3.50
5 slaves	2.19	2.59	3.03	3.48	3.77	4.34	5.83	6.41

Table 1: Speed-up of parallel algorithms, $k = 5$, $n_1 = 24$, $n_2 = 14$.

m	800	1200	1800	2400	2800	3600	5400	7800
3 slaves, 1 blocked	6.88 s	9.33 s	12.27 s	15.32 s	17.39 s	22.60 s	32.53 s	42.03 s
2 slaves	7.02 s	9.19 s	13.58 s	18.16 s	21.57 s	27.49 s	38.11 s	57.62 s
3 slaves	7.12 s	8.90 s	11.47 s	13.14 s	14.90 s	17.11 s	27.91 s	38.77 s

Table 2: cpu-times on a network with homogeneous hardware, $k = 5$, $n_1 = 24$, $n_2 = 14$.

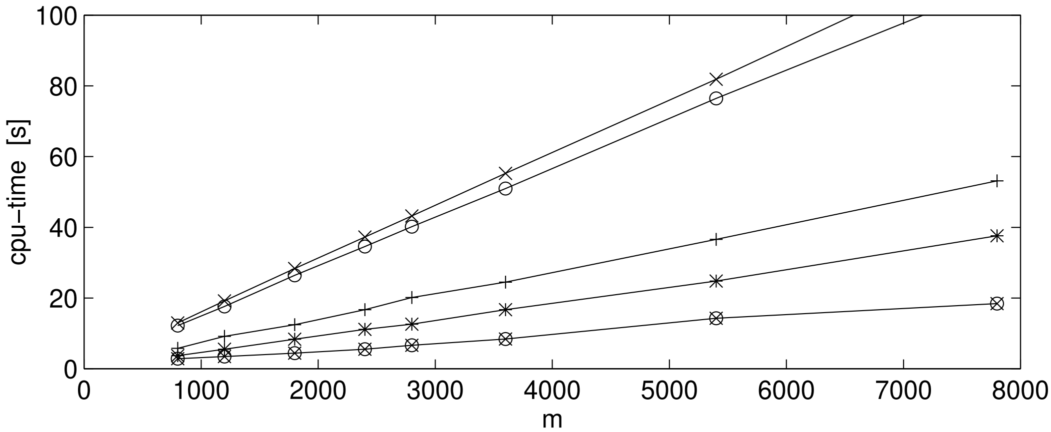

Figure 9: Numerical effort of sequential and parallel algorithms, $k = 3$, $n_1 = 22$, $n_2 = 12$.

The importance of dynamic load balance even on a network with homogeneous hardware is illustrated by the following test:

Example 3 The tests of Example 2 were performed at a daytime when no other users were logged in the workstations (except the `ftp`–server and a few background jobs at the hp 735/99). Now we started the parallel algorithm on the homogeneous network with all four Sparc5's. At the same time a colleague of us did a very cpu-time intensive work on one of the Sparc5's. Thus the corresponding slave slowed down, instead of 67 jobs at each of the slaves we got a typical load balance of 81, 79, and 40 jobs. In Table 2 we summarize the cpu-time for various dimensions m and compare them with the results of Example 2 for the networks with 2 and with 3 slaves. Even with one very slow slave the network with 3 slaves is substantially faster than the one with 2 slaves. Because of dynamic load balance and the asynchronous version of Algorithm II the overall process is not blocked by the slow slave.

Examples 2 and 3 illustrate that the proposed algorithm works satisfactorily in a typical workstation environment. For time-consuming function evaluations f_l the speed-up is proportional to the number of slaves that are added to the network. We want to point out that neither the sequential nor the parallel algorithm is iterative. Except round-off errors the results of both algorithms are identical.

5 Approximated contact conditions in the dynamical simulation

As part of the simulation software the approximation of contact conditions allows the efficient dynamical simulation of wheel-rail systems. In this section we discuss the interaction of approximation and simulation. Simulation results for a simple prototype (rigid wheelset on a straight track) illustrate the efficiency of the proposed algorithm. For the application to real-life problems in railway vehicle industry we refer to [12].

The basic component of a simulation package is the numerical integration of model equations. Advanced packages like SIMPACK combine this basic task with powerful pre- and post-processing facilities to build up the models and to analyse the simulation results ([12]). If approximated contact conditions are used then the user has to specify wheel and rail profile, the coefficients of the approximating spline are computed within the pre-processing step. During the integration of (1) the contact condition (7) has to be evaluated several thousand times (*on-line regularization*). Using the spline approximation the numerical effort is reduced substituting the contact condition by the corresponding spline (*off-line regularization*).

The wheel-rail contact model is only one modul of the MBS model. So we have to avoid complicated data structures and extremely large data arrays in the off-line regularization. Spline approximations are used only in model components that can not be evaluated efficiently on-line. These are the (regularized) geometrical contact condition (7), the contact point S, and all parameters in the computation of friction forces that depend on the curvatures of wheel and rail surface (similar to the definition of S in Section 2 these parameters are defined as weighted mean values along $\mathfrak{C}$).

Remark 3 a) The approximation of the contact condition is part of the constraints $g(q,t)=0$ in (1) and has to be $r+2$ times differentiable to get a r times differentiable right hand side in (2). Therefore the contact condition is approximated by a spline of order k^*+2 and all other parameters (including S) by splines of order k^*, in the actual computations we set $k^* := 3$. All the splines of order k^* are defined on one and the same mesh (i. e. with the same B-splines $B_{kj}^{(i)}$ in (9)) to save cpu-time.

b) The tensorproduct spline was defined such that the approximation gives reasonable results even if it is applied straightforwardly with coordinates $(x^{(1)}, x^{(2)}) = (y_{\mathrm{W}}, \varphi)$ on an equidistant mesh. However, in the simulation package we tune coordinates and meshs to the standard model of wheels with flange and tread. If the flange gets in contact with the rails (at $y_{\mathrm{W}} \approx 3.5\,\mathrm{mm}$ in Figs. 5 and 6) the wheel starts to climb upwards immediately and we have rapid changes in the contact conditions (6) and (7). The jumps of the contact point S from tread to flange of the wheel define a curve in the $(y_{\mathrm{W}}, \varphi)$–plane that can be approximated by a straight line being parallel to the φ-axis in Figs. 1 and 3 ($y_{\mathrm{W}} \approx 3.5\,\mathrm{mm}$). Coordinates $(x^{(1)}, x^{(2)})$ are defined by a slight rotation of the $(y_{\mathrm{W}}, \varphi)$–system such that this line is given by $x^{(1)} = x_*^{(1)}$, i. e. it is parallel to the $x^{(2)}$-axis, $x^{(1)} \approx y_{\mathrm{W}}$, $x^{(2)} \approx \varphi$. The mesh should have more knots in $x^{(1)}$-than in $x^{(2)}$-direction ($n_1 \gg n_2$) since changes of the geometrical data are much less drastic in φ-direction (see Figs. 1 and 2). Starting with an equidistant mesh a few knots in $x^{(1)}$-direction are added in a neighbourhood of $x_*^{(1)}$ to consider the rapid changes of data in this region.

c) After extensive numerical tests the order k of the splines and the dimension $n_1 n_2$ of the spline space are chosen such that the approximation is close to the given data and can nevertheless be evaluated efficiently during the dynamical simulation. We want to keep the data arrays small (using meshs of moderate size i. e. spline spaces of moderate dimensions), excessive work in (9) is avoided if k is not too large. Splines with $n_1 \geq 16 + k$, $n_2 \geq 8 + k$ give reasonable results in the dynamical simulation for wheels with wear profile S1002 and rails with profile UIC60–ORE (both are industrial standards). In the actual computations of Examples 4 and 5 we use $n_1 = 19 + k$, $n_2 = 9 + k$ with $k = k^* + 2$ and $k = k^*$, respectively ($k^* = 3$), i. e. 20×10 rectangles in the mesh.

We illustrate the performance of the rigid contact models with results for 2 benchmark problems that consider the motion of a rigid wheelset on a straight track. For both benchmarks the flanges of the wheels do not get in contact with the rails. The basic contact model (6) introduces for both benchmarks discontinuities in the solution ([10]) and is not considered here. We compare on-line and off-line regularization for the regularized contact model (7) with $\nu = 2.0_{10} - 5$, ($s_i - s_{i-1} \approx 0.5\,\mathrm{mm}$ at the tread, $s_i - s_{i-1} \approx 0.2\,\mathrm{mm}$ at the flange of the wheel). In the simulation the friction forces are computed according to Kalker's theory applied on an equivalent Hertzian contact patch ([9]). The model equations (1) are integrated in Gear–Gupta–Leimkuhler formulation using the BDF code DASSL ([2]).

Example 4 It is a well known phenomenon that the motion of a rigid wheelset at a straight track gets unstable if its speed exceeds a so-called critical speed. The quasi-periodic solution approaches a limit cycle. Fig. 10 shows the phase plot $(y(t), \dot{y}(t))$ for the lateral displacement y of the wheelset, $y(t_0) = 4.5\,\mathrm{mm}$. The left plot corresponds to the on-line evaluation of (7), the right one to the spline approximation. In the real physical system this limit cycle is typical and robust against changes of system parameters. The solution trajectory, however, is very sensitive w. r. t. small changes in the system. Both simulation results reproduce the limit cycle and are in good agreement with measured data. They differ, however, in the trajectories: with approximated contact conditions (right plot in Fig. 10) the solution approaches the limit cycle substantially faster. Varying the regularization parameter ν in (7) we found that this difference in the local behaviour of the solution is in the range of the model error that is caused by the rigid contact model. From the point of view of dynamical simulation the off-line regularization gives satisfactory results. Its main benefit is a reduction of cpu-time by a factor of 11. In Fig. 10 the motion was simulated on a time interval of 10.0 s with 624.8 s cpu-time for the on-line regularization and 56.3 s cpu-time for the simulation using the approximated contact conditions.

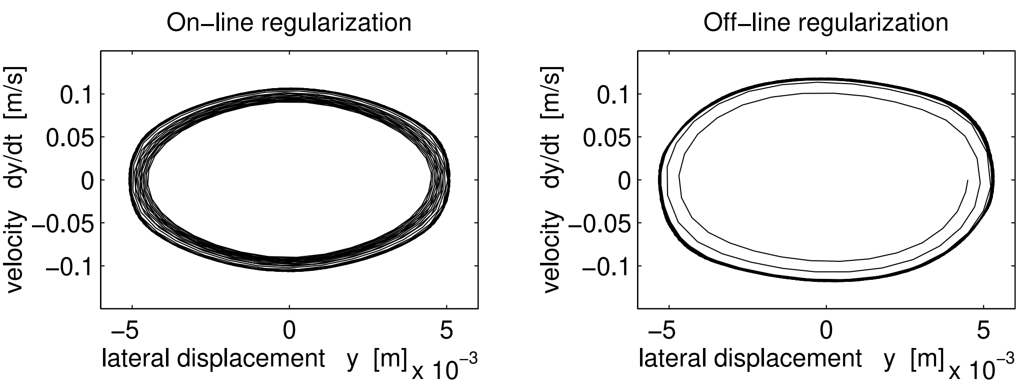

Figure 10: Rigid wheelset moving on a straight track. Limit cycles computed with on-line (left) and off-line (right) regularization.

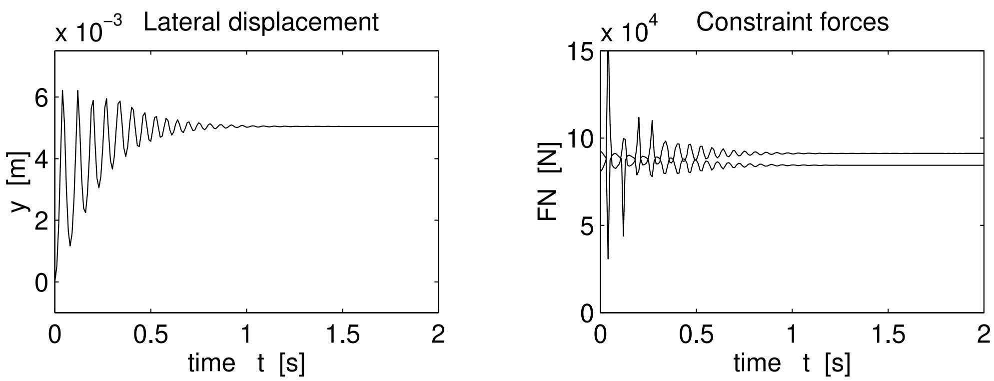

Figure 11: Simulation results for the regularized contact model applied to the benchmark of Pascal/Sauvage ([11]).

Example 5 In simulation packages it has been popular to consider only the local contact geometry. This simplifies substantially the computation of friction forces but results in large model errors in the neighbourhood of jumps of the contact point in Fig. 1. A benchmark in [11] was constructed such that these models fail totally (e. g. [10]). A force $F_y = 20\,\mathrm{kN}$ acting in lateral direction shifts the wheelset to the right ($y \approx 5.0\,\mathrm{mm}$). The motion tends to a quasi-stationary state that is at the right wheel very close to a configuration for that the contact point in the basic contact model (6) jumps. For the regularized contact model (7) the numerical solution reaches both for on-line and off-line regularization this quasi-stationary state, Fig. 11 shows in the left plot the vertical displacement of the wheelset and in the right plot the constraint forces on left and right wheel. The differences between the simulation results with on-line and with off-line regularization are negligible. Using the spline approximation the cpu-time is, however, reduced from 497.1 s to 42.4 s, i. e. by the factor 12.

Remark 4 a) The drastic reduction of cpu-time makes approximated contact conditions very attractive for simulation packages. In SIMPACK approximation is used as default contact model whenever it is applicable. The on-line evaluation of the regularized contact condition (7) is restricted to simulations with rail profiles that vary along the track (e. g. a wheelset running through a switch).

b) If parts of the model equations (1) are substituted by spline functions then classical theory

(e. g. [8, Chapter II.3]) predicts an order reduction for higher order integrators because of the discontinuities in higher derivatives of the spline. In our numerical tests the stepsize of time integration was not reduced if the on-line evaluation of (7) was substituted by the spline function (both for BDF and Runge–Kutta codes). I. e. the use of spline approximations did not affect the efficiency of the integration software (similar observations are known from other applied problems [7]).

Summary

The complicated geometry of the contact between wheels with wear profiles and rails causes severe problems in the dynamical simulation of wheel-rail systems. Realistic models for contact geometry and friction forces yield to a very large numerical effort in the integration of the model equations. For a great variety of standard problems the dynamical simulation can be accelerated by a factor of 10 using *approximations* of geometrical data during the dynamical simulation. We propose a tensorproduct spline approximation that is computed in the pre-processing of the integration. Using orthogonal transformations the spline coefficients are obtained as least squares solution of a large sparse overdetermined system of linear equations. The numerical effort in the pre-processing is kept small exploiting the sparsity structure of this system, further improvements are achieved by parallelization. If the spline approximation is used then benchmark problems for simple prototypes and real-life applications from railway industry are solved satisfactorily.

The computation of the tensorproduct spline was implemented in FORTRAN. The sequential version of this code WRSPL (**W**heel–**R**ail–**SPL**ine) is available freely for non-commercial use and can be obtained contacting the first author. It is integrated in the commercial MBS simulation package SIMPACK that is tailored to industrial applications.

Acknowledgement Part of this work has been done within the joint project "Differential equations and singular manifolds in the dynamical simulation of wheel-rail systems" of Rostock University and DLR Oberpfaffenhofen that is supported by the German Federal Minister for Education, Science, Research, and Technology (grant FR7ROK). We thank our colleagues in the wheel-rail groups in Rostock and Oberpfaffenhofen for very stimulating discussions about the topic of the present paper. The authors are especially grateful to Dirk Langemann (Rostock) for his help in the implementation of the smoothing term.

References

[1] M. Arnold, K. Frischmuth, M. Hänler, and H. Netter. Differentialgleichungen und singuläre Mannigfaltigkeiten in der dynamischen Simulation von Rad–Schiene–Systemen. In N.N., editor, *N.N.*, Springer-Verlag, Berlin Heidelberg New York, 1996.

[2] K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical solution of initial–value problems in differential–algebraic equations.* SIAM, Philadelphia, 2nd edition, 1996.

[3] C. de Boor. *A practical guide to splines.* Springer-Verlag, New York, 1978.

[4] P. Dierckx. *Curve and surface fitting with splines.* Clarendon Press, Oxford New York Tokyo, 1993.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM3 User's Guide and Reference Manual. Technical Report ORNL/TM–12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee, September 1994.

[6] G.H. Golub and Ch.F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore London, 2nd edition, 1989.

[7] E. Hairer. private communication, 1996.

[8] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations. I. Nonstiff Problems.* Springer-Verlag, Berlin Heidelberg New York, 2nd edition, 1993.

[9] J.J. Kalker. *Three-Dimensional Elastic Bodies in Rolling Contact.* Kluwer Academic Publishers, Dordrecht Boston London, 1990.

[10] H. Netter and M. Arnold. Geometrie und Dynamik eines Rad-Schiene-Modells in Deskriptorform mit unstetigen Zustandsgrößen. Technical Report IB 515–93–02, DLR, D-5000 Köln 90, 1993.

[11] J.P. Pascal. Benchmark to test wheel/rail contact forces. Technical report, INRETS Paris, 1990.

[12] W. Rulka, A. Haigermoser, L. Mauer, and H. Netter. Anwendung moderner Auslegungsstrategien für Schienenfahrzeuge durch Einsatz des Simulationsprogramms SIMPACK. VDI Berichte Nr. 1219, Düsseldorf, 1995.

[13] H. Schwetlick and V. Kunert. Spline smoothing under constraints on derivatives. *BIT*, 33:512–528, 1993.

[14] B. Simeon, C. Führer, and P. Rentrop. Differential-algebraic equations in vehicle system dynamics. *Surveys on Mathematics for Industry*, 1:1–37, 1991.