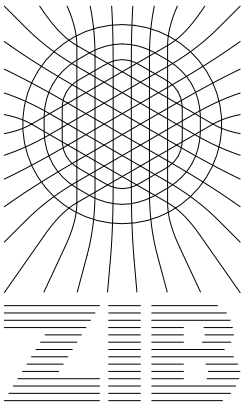
Konrad-Zuse-Zentrum für Informationstechnik Berlin



Ch. Lubich* U. Nowak U. Pöhle
Ch. Engstler**

MEXX – Numerical Software for the Integration of Constrained Mechanical Multibody Systems

Preprint SC 92–12 (December 1992)

Ch. Lubich* U. Nowak U. Pöhle
Ch. Engstler**

MEXX – Numerical Software for the Integration of Constrained Mechanical Multibody Systems

Abstract. MEXX (short for MEXanical systems eXtrapolation integrator) is a Fortran code for time integration of constrained mechanical systems. MEXX is suited for direct integration of the equations of motion in descriptor form. It is based on extrapolation of a time stepping method that is explicit in the differential equations and linearly implicit in the nonlinear constraints. It only requires the solution of well–structured systems of linear equations which can be solved with a computational work growing linearly with the number of bodies, in the case of multibody systems with few closed kinematic loops. Position and velocity constraints are enforced throughout the integration interval, whereas acceleration constraints need not be formulated. MEXX has options for time–continuous solution representation (useful for graphics) and for the location of events such as impacts. The present article describes MEXX and its underlying concepts.

* Universität Würzburg ** Universität Innsbruck

This work was supported in part by a Konrad Zuse Fellowship, and by the Austrian Science Foundation project P8443–PHY.

Contents

Introduction

The dynamic simulation of constrained mechanical systems is of importance in application areas such as robotics, vehicle and machinery design, and biomechanics, see e.g. [23, 29, 31]. The traditional approach to formulating and numerically solving the equations of motion of multibody systems has been to use a suitably chosen set of minimal coordinates, in terms of which the equations of motion become a second–order system of ordinary differential equations. These are in turn integrated by standard ODE solvers. More recently, progress in numerical solution techniques for differential–algebraic equations and the comparative ease of building up descriptor formulations of mechanical systems (i.e., formulations in non–minimal sets of coordinates) have spurred the development of numerical methods which directly solve equations of motion in descriptor form, see e.g. [1, 5, 7, 14, 21, 24, 28, 34]. Methods which automatically determine a local set of minimal coordinates during integration, such as those based on generalized coordinate partitioning [36] are widely used in commercial multibody software packages and are in some respects quite close to differential–algebraic solvers, but certainly differ in that they are less versatile in handling the linear algebra.

The present paper describes the Fortran code MEXX (short for MEXanical systems eXtrapolation integrator) and its underlying concepts. MEXX is suited for direct integration of the equations of motion in descriptor form, and has the following features:

- Time stepping by a half–explicit extrapolation method, allowing for the accurate and robust computation of position, velocity, acceleration, and constraint forces.
- Only position and velocity constraint functions are evaluated, acceleration constraints need not be formulated.
- Both position and velocity constraints remain satisfied throughout the integration interval.
- Uses well–structured linear algebra, enabling the use of $O(n)$ recursive elimination, among other full and sparse linear algebra options.
- Time–continuous solution representation (e.g. for graphics)
- Root–finding options (e.g. for unilateral constraints and Coulomb friction problems)

MEXX encourages the use of large, sparse descriptor formulations, but can also efficiently handle near–statespace kinematic formulations of multibody systems.

In Section 1 we discuss the class of equations that are to be solved by MEXX. Section 2 describes discretization and related issues, and Section 3 deals with various linear algebra options. Section 4 gives a survey of implementation aspects. Finally, in Section 5 some numerical experiments illustrate the performance of MEXX.

Chapter 1

Problem formulation

1.1 Equations of motion

MEXX is to solve initial value problems for nonstiff equations of motion in descriptor form. The equations for position $p(t)$, velocity $v(t)$, Lagrange multipliers $\lambda(t)$, and (optionally) external dynamics variables $u(t)$, take the form

$$\begin{array}{lrcl} \text{a)} & \dot{p} & = & T(t,p)v \\ \text{b)} & M(t,p)\dot{v} & = & f(t,p,v,\lambda,u) - G(t,p)^T\lambda \\ \text{c)} & 0 & = & G(t,p)\cdot v + g^I(t,p) \\ \text{d)} & \dot{u} & = & d(t,p,v,\lambda,u) \end{array} \tag{1.1}$$

with position constraints treated as invariants:

$$0 = g(t,p)\ , \tag{1.1e}$$

and with prescribed initial values

$$p(t_0) = p_0,\ v(t_0) = v_0,\ u(t_0) = u_0\ . \tag{1.2a}$$

If f depends non–linearly on λ, then also an approximation of

$$\lambda(t_0) = \lambda_0 \tag{1.2b}$$

should be specified. The dimension and interpretation of the unknown variables are the following:

$p \in \mathbb{R}^{n_p}$	position variables	
$v \in \mathbb{R}^{n_v}$	velocity variables	$(n_v \le n_p)$
$\lambda \in \mathbb{R}^{n_\lambda}$	Lagrange multipliers	$(n_\lambda \le n_v)$
$u \in \mathbb{R}^{n_u}$	variables of external dynamics	

The functions in (1.1) are assumed to be sufficiently smooth, with the fol-

lowing (obvious) dimensions:

$$\begin{aligned}
T &: \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_p \times n_v} \\
M &: \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_v \times n_v} \\
G &: \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_\lambda \times n_v} \\
f &: \mathbb{R} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_v} \times \mathbb{R}^{n_\lambda} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_v} \\
d &: \mathbb{R} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_v} \times \mathbb{R}^{n_\lambda} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_u} \\
g^I &: \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_\lambda} \\
g &: \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_g}
\end{aligned}$$

We assume that for all (t, p) in a neighbourhood of the solution the following conditions are satisfied:

$$M(t,p) \text{ is symmetric and positive semi–definite .} \tag{1.3a}$$

$$G(t,p) \text{ has full row rank .} \tag{1.3b}$$

$$a^T M(t,p) a > 0 \text{ for all } a \in \mathbb{R}^{n_v} \text{ with } G(t,p)a = 0 \,. \tag{1.3c}$$

If (1.3a) holds, then (1.3b,c) are equivalent to:

$$\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \text{ is invertible .} \tag{1.4a}$$

This assumption is sufficient as long as the Jacobian

$$F := \frac{\partial f}{\partial \lambda}$$

is "small" in a neighbourhood of the solution, which is satisfied in many applications. We note that, physically, the dependence of f on λ models the effect of *dry friction* in the joints of the mechanical system.

Otherwise, we have to require the following condition instead of (1.4a):

$$\begin{pmatrix} M & G^T - F \\ G & 0 \end{pmatrix} \text{ is invertible along the solution .} \tag{1.4b}$$

Under these assumptions, (1.1) is a differential–algebraic system of index 2, and local existence and uniqueness of the solution are guaranteed [7, 19, 21]. Equation (1.1) encompasses a variety of mechanical formulations. The equations of motion of a single multibody system can be formulated in many

different ways within the framework of (1.1): formulations in absolute (body) coordinates with joints modeled by constraints, formulations using relative (joint) coordinates, or mixed formulations. We refer to [23, 29, 31, 37] for various possibilities, see also [1, 6, 24, 35]. Here we include a brief discussion which should serve to clarify the possible physical meanings of the functions and variables in (1.1). We consider a mechanical systems of n_B rigid bodies interconnected by n_J joints and by force elements.

1.2 Absolute coordinates

(Implicit joint formulation, see in particular [23])

In a *planar* mechanical system, the position of body "k" ($k = 1, \ldots, n_B$) is determined by the Cartesian coordinates of its center of gravity (x_k, y_k), and by a rotation angle φ_k. Here we have position variables

$$p = (\vec{p_k})_{k=1}^{n_B}, \quad \text{with } \vec{p_k} = (x_k, y_k, \varphi_k) \ . \tag{1.5}$$

Velocity variables are

$$\dot{p} = v$$

and we have $n_p = n_v = 3n_B$. The Newton–Euler equations of motion are then (1.1b) with the constant diagonal matrix

$$M = \text{ blockdiag } (M_k) \text{ with } M_k = \begin{pmatrix} m_k & 0 & 0 \\ 0 & m_k & 0 \\ 0 & 0 & I_k \end{pmatrix} (k = 1, \ldots, n_B) \tag{1.6}$$

where m_k is the mass of body k, and I_k its inertia. The right–hand side vector f in (1.1b) is composed of the corresponding applied forces and torques. The joints imply position constraints (1.1e) with

$$g = (\vec{g_j})_{j=1}^{n_J} \text{ where } \vec{g_j}(t, p) = \vec{g_j}(t, \vec{p_k}, \vec{p_l}) \ , \tag{1.7}$$

if joint j is connecting bodies k and l. The number of components in $\vec{g_j}$ equals 3 minus the number of degrees of freedom of joint j. The velocity constraint equations are obtained by differentiating (1.1e) with respect to time, yielding (1.1c) with

$$G = \frac{\partial g}{\partial p}, \ g^I = \frac{\partial g}{\partial t} \ . \tag{1.8}$$

Note that $g^I \equiv 0$ if there is no kinematic excitation in the joints. By the d'Alembert–Lagrange principle, the reaction forces in (1.1b) are orthogonal to the nullspace of G, hence of the form $r = G^T \lambda$.

In a *spatial* mechanical system, the situation is complicated by the fact that angular velocity is not integrable. The orientation of a body is usually described by 3 angles (Eulerian angles, or Tait–Bryan angles) or by 4 Euler parameters. These taken together with the Cartesian coordinates of the center of gravity form the position vector p. On the other hand, it is convenient to form the velocity vector v from the velocities of the centers of gravity and the angular velocities of the bodies. Then the Newton–Euler equations give (1.1b) with a constant, block–diagonal matrix M, whose blocks consist of the mass times the 3×3 identity matrix and the 3×3 inertia tensors. However,

position and velocity variables are then related by (1.1a) where $T(t,p)$ is no longer the identity. In the case of a formulation with Eulerian or Tait–Bryan angles, T has 3×3 blocks which relate the time derivatives of these angles to angular velocity. Here $n_p = n_v = 6n_B$. In a formulation with Euler parameters, T is no longer square but is block diagonal with 4×3 blocks and 3×3 identity matrices on the diagonal. Here $n_p = 7n_B$ and $n_v = 6n_B$. The normalization constraint of the Euler parameters forms part of the position constraints (1.1e), whose remaining components are of a form as in (1.7). The velocity constraints (1.1c) can again be obtained from differentiation of (1.1e) (and using (1.1a)), in which case one gets (1.1c) with

$$G = \frac{\partial g}{\partial p} \cdot T, \; g^I = \frac{\partial g}{\partial t} \; . \tag{1.9}$$

Especially if $\frac{\partial g}{\partial t} \equiv 0$ (no kinematic excitation), it is often more efficient to formulate the velocity constraints directly from kinematic considerations. This yields (1.1c) with a matrix G which has the same null–space as $G = \frac{\partial g}{\partial p} \cdot T$, but is not necessarily identical to it. This also changes the Lagrange multipliers λ, but leaves $r = G^T \lambda$ invariant (both for the exact and the numerical solution).

1.3 Relative coordinates

(Explicit joint formulation).

Traditionally, the use of relative (joint) coordinates has been to reduce the dimension of the system, leading to a formulation of the equations of motion as an ODE system of minimal dimension, at least in the case of tree–configured systems. For systems with closed kinematic loops, the loop–closing constraints might be left as constraints, leading to a system (1.1) with matrices M and G which are full and usually depend on the variables in a complicated way, but which is of reduced dimension as compared to the previous subsection. From a computational viewpoint, the reduction process to minimal (or near–minimal) dimension can be problematic for large systems for two reasons:

- Unless there are only few degrees of freedom left in the system, the loss of sparsity can make linear algebra computations much more expensive for the reduced system.
- The evaluation of the right–hand side in (1.1b) for the reduced system requires the evaluation of generalized gyroscopic forces, which may be costly to compute.

The use of joint coordinates is nevertheless attractive for the kinematic formulation of joints which have only 1 or 2 degrees of freedom. Consider for example an elbow joint whose state is definitely easier to describe by one angle rather than by five constraints.

The above mentioned difficulties can be avoided if one uses joint coordinates together with body coordinates, without explicitly performing the reduction process. This leads to a system (1.1) in the following way, cf. [24, 35]. Let p denote absolute coordinates of the system in the same way as in Section 1.2, and let $q = (\vec{q_j})_{j=1}^{n_J}$ denote joint coordinates where $\vec{q_j}$ is composed of the degrees of freedom of joint j. In the case of a tree–configured system, p can be expressed in terms of q. This relation is generally implicit in p:

$$k(p, q, t) = 0 \,, \tag{1.10}$$

where $\partial k/\partial p$ is invertible, and $\partial k/\partial q$ has full column rank. Let v denote the velocity variables, related to the derivatives of the absolute position variables by (1.1a), where for simplicity we now assume $T(t, p)$ to be square and invertible (thus excluding Euler parameters for the moment). Differentiating (1.10) and using (1.1a) gives

$$Kv + J\dot{q} + \frac{\partial k}{\partial t} = 0 \,, \tag{1.11}$$

where we have denoted

$$K = \frac{\partial k}{\partial p} \cdot T, \; J = \frac{\partial k}{\partial q} \,. \tag{1.12}$$

By d'Alembert's principle, the dynamic equations of motion are

$$M\dot{v} - f = r \;\; \text{with } (K^{-1}J)^T r = 0 \,. \tag{1.13}$$

Typically, K and J are sparse matrices, but K^{-1} is not. To avoid expressions with K^{-1}, we introduce μ by setting $r = K^T\mu$. Then $J^T\mu = 0$, and (1.13) can be written as

$$\begin{pmatrix} M & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{v} \\ \ddot{q} \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} - \begin{pmatrix} K^T \\ J^T \end{pmatrix} \mu \,. \tag{1.14}$$

If we put

$$\begin{aligned} &\boldsymbol{p} = \begin{pmatrix} p \\ q \end{pmatrix}, \; \boldsymbol{v} = \begin{pmatrix} v \\ \dot{q} \end{pmatrix}, \; \boldsymbol{\lambda} = \mu \\ &\boldsymbol{M} = \begin{pmatrix} M & 0 \\ 0 & 0 \end{pmatrix}, \; \boldsymbol{G} = (K, J), \; \boldsymbol{T} = \begin{pmatrix} T & 0 \\ 0 & I \end{pmatrix}, \; \boldsymbol{g} = k \end{aligned} \tag{1.15}$$

then the equations (1.14), (1.11), (1.10) are exactly of the form (1.1) in bold letters. This DAE formulation does not suffer from the difficulties with large systems mentioned in the beginning of this subsection, yet makes the potential advantages of using joint coordinates in the formulation of the kinematic equations accessible. We remark that the reduction process to small dimension, when it is efficient, can still be mimicked in the linear algebra to be used in the numerical solution of the augmented system considered here, see Section 3.1.

In the presence of closed kinematic loops, loop–closing constraints have yet to be added to the above system. This again leads to a system of the form (1.1).

1.4 Further remarks on (1.1)

Equation (1.1) also admits the treatment of nonholonomic constraints, which lead to $n_g < n_\lambda$. The variable u is an additional dynamic variable, needed for modeling non–mechanical elements interacting with the multibody system (e.g. a motor driving a joint) or a control variable. A system of the form (1.1) is also obtained with elastic instead of rigid bodies, after discrete modelization of elastic components.

Chapter 2

Discretization

In this section we describe the extrapolation method implemented in MEXX. This is a variant of a method in [24].

2.1 Basic discretization

To get at $t_n = t_0 + nh$ approximations

$$p_n \approx p(t_n),\ v_n \approx v(t_n),\ a_n \approx \dot{v}(t_n),\ \lambda_n \approx \lambda(t_n),\ u_n \approx u(t_n)\ ,$$

we have chosen the following half–explicit Euler method:

$$p_{n+1} = p_n + h\dot{p}_n,\ \dot{p}_n = T(t_n, p_n)v_n \tag{2.1a}$$

$$\begin{pmatrix} M_{n+1} & G_{n+1}^T \\ G_{n+1} & 0 \end{pmatrix} \begin{pmatrix} v_{n+1} \\ h\lambda_{n+1} \end{pmatrix} = \begin{pmatrix} M_{n+1}v_n + hf_n \\ -g_{n+1}^I \end{pmatrix} \tag{2.1b}$$

$$a_{n+1} = (v_{n+1} - v_n)/h \tag{2.1c}$$

$$u_{n+1} = u_n + h\dot{u}_n,\ \dot{u}_n = d(t_{n+1}, p_{n+1}, v_{n+1}, \lambda_{n+1}, u_n) \tag{2.1d}$$

with $M_n = (t_n, p_n),\ G_n = G(t_n, p_n),\ f_n = f(t_n, p_n, v_n, \lambda_n, u_n),\ g_n^I = g^I(t_n, p_n)$. We note that all steps in (2.1) are explicit, with exception of the solution of the linear system (2.1b) which will be discussed in Section 3. The matrix is invertible by (1.4a).

The above discretization is to be used as long as

$$\|F\| \ll 1, \quad \text{where } F = \frac{\partial f}{\partial \lambda}$$

that is, unless there is substantial dry friction in the system. If F is large, then the linear system (2.1b) is replaced by (cf. (1.4b))

$$\begin{pmatrix} M_{n+1} & G_{n+1}^T - F_0 \\ G_{n+1} & 0 \end{pmatrix} \begin{pmatrix} v_{n+1} \\ h\lambda_{n+1} \end{pmatrix} = \begin{pmatrix} M_{n+1}v_n + hf_n - F_0 \cdot h\lambda_n \\ -g_{n+1}^I \end{pmatrix} \tag{2.2}$$

where F_0 is $\partial f/\partial \lambda$ evaluated at the starting values of the extrapolation step. This modified system is more costly to solve. The Jacobian F_0 has to be supplied, and the symmetry of the matrix is lost.

2.2 Extrapolation

We denote the numerical solution of method (2.1) by

$$x_n = (p_n, v_n, a_n, \lambda_n, u_n)^T \ . \tag{2.3}$$

If $\partial f/\partial\lambda \equiv 0$, then there exists an unperturbed h–expansion of the error:

$$x_n - x(t_n) = he_1(t_n) + h^2 e_2(t_n) + \ldots + h^N e_N(t_n) + O(h^{N+1}) \ . \tag{2.4}$$

Otherwise, there exists a perturbed h–expansion

$$x_n - x(t_n) = h\left(e_1(t_n) + \varepsilon_n^{(1)}\right) + \ldots + h^N(e_N(t_n) + \varepsilon_n^{(N)}) + O(h^{N+1}) \tag{2.5a}$$

where $\varepsilon_n^{(j)}$ are independent of h and decay geometrically, bounded by

$$\|\varepsilon_n^{(j)}\| \le C \cdot \rho^n \tag{2.5b}$$

where, roughly, ρ is proportional to the size of $F = \partial f/\partial\lambda$. The modified scheme (2.2) also has an unperturbed h–expansion (2.4). Detailed proofs of these expansions are given in [15], cf. also [19].

Richardson extrapolation (see e.g. [8, 20]) successively eliminates the error terms $e_j(t)$, and it also effectively reduces the error caused by geometrically decaying perturbation terms $\varepsilon_n^{(j)}$, cf. [18]. The algorithmic description is as follows: Given a basic step size H, one constructs approximations to $x(t_0+H)$ using the discretization method (2.1) (or (2.2), if necessary) with step sizes $h_j = H/n_j$, where $\{n_j\} = \{2, 3, 4, 5, 6, 7, 8, 10, 12, \ldots\}$ is the step number sequence. We denote by $x(t_0 + H, h_j) = x_{n_j}$ the approximation obtained with step size h_j. The h–extrapolation tableau is given by the formulas

$$\begin{aligned} T_{j,1} &= x(t_0 + H, h_j) \\ T_{j,k+1} &= T_{j,k} + \frac{T_{jk} - T_{j-1,k}}{(n_j/n_{j-k}) - 1} \ , \quad k+1 \le j \ . \end{aligned} \tag{2.6}$$

Error estimates are obtained for all solution components from the subdiagonal differences $T_{j,j} - T_{j,j-1}$ [8]. An adaptive order and step size control is based on the error estimates for the variables p, v, and u using the scaled norms

$$\|(\Delta p, \Delta v, \Delta u)^T\|_{\mathrm{TOL}} = \sqrt{\|\Delta p\|^2_{\mathrm{TOL}} + \|\Delta v\|^2_{\mathrm{TOL}} + \|\Delta u\|^2_{\mathrm{TOL}}} \tag{2.7}$$

where

$$\|\Delta p\|^2_{\mathrm{TOL}} = \frac{1}{n_p} \sum_{i=1}^{n_p} \left(\frac{\Delta p_i}{w_i}\right)^2 \tag{2.8a}$$

and, essentially,

$$w_i = \mathrm{RTOL}_i \cdot |p_i| + \mathrm{ATOL}_i \ . \tag{2.8b}$$

The corresponding norms for Δv and Δu are defined analogously. The "index–2 variables" a and λ are not used for error control, cf. [27, 19]. Our implementation of the error and stepsize control is patterned after that of the code ODEX in [20].

The computational cost for one basic integration step, using (2.1) and j lines in the extrapolation tableau, is thus as follows:

number of $M, G, G^I -$ evaluations: $n_1 + n_2 + \ldots + n_j$
number of $f, \dot{p}, \dot{u} -$ evaluations: $1 + (n_1 - 1) + \ldots + (n_j - 1)$
linear systems (decomposition and solution): $n_1 + n_2 + \ldots + n_j$
number of matrix–vector multiplications with M : $2(n_1 + \ldots + n_j)$.

In addition, there is a number of vector operations in the discretization scheme and in building the extrapolation tableau.

2.3 Projection

An approximation x^0 of $x(t)$ obtained by extrapolation does not, in general, satisfy the position and velocity constraints. After every successful extrapolation step, we therefore include a projection onto the constraint manifold.

(a) Projection onto $g(t,p) = 0$: Let an approximation p^0 of $p(t)$ be given. This is projected via

$$\begin{aligned} p &= p^0 + T_0 \cdot \nu \\ M_0\nu + C_0^T\mu &= 0 \\ 0 &= g(t,p) \ . \end{aligned} \tag{2.9}$$

Here (p, ν, μ) are the unknowns, and

$$T_0 = T(t,p^0),\ M_0 = M(t,p^0),\ C_0 = C(t,p^0)$$

with

$$C = \frac{\partial g}{\partial p} \cdot T \ . \tag{2.10}$$

Note that $C = G$ in many formulations, cf. Section 1. We remark that this special projection is invariant under affine coordinate transformations, $\tilde{p} = Ap + a$. The nonlinear system (2.9) is solved by modified Newton iterations:

$$\begin{pmatrix} M_0 & C_0^T \\ C_0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta\nu^k \\ \mu^{k+1} \end{pmatrix} = - \begin{pmatrix} M_0\nu^k \\ g(t,p^k) \end{pmatrix} \tag{2.11a}$$

$$\begin{aligned} \nu^{k+1} &= \nu^k + \Delta\nu^k \ , \ \ \nu^0 = 0 \\ p^{k+1} &= p^k + T_0 \Delta\nu^k \end{aligned} \tag{2.11b}$$

until $\|T_0\Delta\nu^k\|_{\mathrm{TOL}} \leq 10^{-2}$ with the scaled norm of (2.8). Then we accept $p = p^{k+1}$.

(b) Projection onto $G(t,p)v + g^I(t,p) = 0$: An approximation v^0 of $v(t)$ is projected via the linear system

$$\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} v \\ \mu \end{pmatrix} = \begin{pmatrix} Mv^0 \\ -g^I \end{pmatrix} \tag{2.12}$$

with $M = M(t,p),\ G = G(t,p),\ g^I = g^I(t,p)$. This projection is again invariant under affine transformations of variables, $\tilde{v} = Av + a$.

We emphasize that (2.9) is the only nonlinear system which appears in the overall integration algorithm. Projection is done only after every complete extrapolation step. Moreover, p^0 is an approximation whose error has been found to be less than the prescribed error tolerance by the local error control of the extrapolation method. Therefore 1 or at most 2 iterations in (2.11) are usually sufficient.

The projections (2.9) and (2.12) are also done in the very first step before the integration is started, in order to ensure consistent initial values. MEXX does not start from "highly inconsistent" initial data which are such that (2.11) does not converge.

2.4 Dense output

Extrapolation does not only provide accurate solution approximations at the grid points $t_0, t_0 + H$, etc., but can further be used to yield a continuous approximation of about the same accuracy over the whole integration interval. This is used in the root–finding option of MEXX, and is of course also important for the graphical representation of the solution. The basic idea here is to first compute accurate approximations of solution derivatives at the endpoints of the interval $[t_0, t_0 + H]$ by extrapolation of divided differences of solution approximations (2.1), and then to construct a polynomial having the same endpoint values and derivatives. Approximation properties and implementation of this approach have been studied in [22]. For mechanical systems (1.1), the algorithm reads as follows.

We split the solution vector into its "differential" and "algebraic" components:

$$y = \begin{pmatrix} p \\ v \\ u \end{pmatrix} \ , \ \ z = \begin{pmatrix} a \\ \lambda \end{pmatrix} \ . \tag{2.13}$$

In the very first step, a starting value for y is readily available (using the projected values of p_0 and v_0 according to Section 2.3, and u_0 as given). However, a starting value of z is not known a priori. When the first extrapolation step is completed, we obtain consistent approximate values of both y and z at $t_0 + H$. An approximation of $z(t_0)$ is obtained by h–extrapolation of the values z_1 of (2.1) already computed with the step sizes of the extrapolation tableau. For the differential components we then also get the derivative values $\dot{y}$ at both endpoints, using a function evaluation for $\dot{p}$ and $\dot{u}$ at $t_0 + H$ which anyway has to be done in the next extrapolation step. We are thus left with the task of finding accurate approximations of

$$\ddot{y}, y^{(3)}, y^{(4)}, \ldots$$
$$\dot{z}, \ddot{z}, z^{(3)}, \ldots$$

at both endpoints.

Let us assume that κ lines of the extrapolation tableau (2.6) have been computed in the step, using step size $h_1 > \ldots > h_\kappa$ with $h_j = H/n_j$, and $n_1 \geq 2$. At the left endpoint we form the divided forward differences

$$l_j^{(k)} = \frac{1}{h_j^k} \begin{pmatrix} \Delta^k \dot{y}_0 \\ \Delta^k z_1 \end{pmatrix} \text{ for } k = 1, \ldots, \lambda \leq \kappa \text{ and } j = k, \ldots, \kappa \tag{2.14}$$

where Δ denotes the forward difference operator ($\Delta x_n = x_{n+1} - x_n$), and $\dot{y}_n = (y_{n+1} - y_n)/h$. In (2.14), we have not indicated the obvious dependence on the step size h_j in the notation of $\Delta^k \dot{y}_0$ and $\Delta^k z_1$. Similarly, at the right endpoint we use the divided backward differences

$$r_j^{(k)} = \frac{1}{h_j^k} \begin{pmatrix} \nabla^k \dot{y}_{n_j - 1} \\ \nabla^k z_{n_j} \end{pmatrix} \text{ for } k = 1, \ldots, \rho \leq \kappa \text{ and } j = k, \ldots, \kappa \,. \tag{2.15}$$

Our choice of λ and ρ is such that $\lambda + \rho = \kappa - 1$, and $\rho - 1 \leq \lambda \leq \rho$. These divided differences represent approximations of the solution derivatives $y^{(k+1)}$ and $z^{(k)}$ at the endpoints and again have an asymptotic h–expansion. Extrapolating $(\kappa - k)$–times therefore gives improved approximations $l^{(k)}$ and $r^{(k)}$ of the derivatives of $\dot{y}$ and z at the endpoints of the interval. With the help of Newton's interpolation formula, we then construct a polynomial Y of degree $\lambda + \rho$ whose derivatives at the endpoints coincide with the approximate solution derivatives $y, \dot{y}$, and $y^{(k+1)}$ as computed above, and the polynomial Z of degree $\lambda + \rho - 1$ with endpoint derivatives $z, z^{(k)}$ as computed above.

2.5 Event location

MEXX has an option for locating zeros of switching functions

$$\phi_i(t, p(t), v(t), a(t), \lambda(t), u(t)) = 0 \quad \text{for some} \quad i = 1, \ldots, s \ . \tag{2.16}$$

This uses the dense output and the projection of the foregoing sections. If at least for one index i, the sign of ϕ_i is not the same at both endpoints of an integration interval $[t_0, t_0 + H]$, then a special scalar Newton method is used to find the zeros t_i in the interval. First, heuristically damped Newton iterations are started, taking the values of the dense output as arguments in ϕ_i. These t_i are then taken as starting iterates in modified Newton methods for $\phi_i = 0$ which now use the *projected* dense output (with projections as in Section 2.3) as solution approximation in the evaluations of ϕ_i. Depending on the user's choice, MEXX either stops the current integration at the left–most t_i or continues the integration at $t_0 + H$ after reporting all zeros t_i.

Chapter 3

Linear algebra

The integration method (2.1) requires in every step the solution of a linear system

$$\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} w \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{3.1}$$

where M and G satisfy (1.3). Linear systems of this form arise in a variety of applications, notably in constrained least squares [3] and constrained optimization [16]. A variety of algorithms has been proposed for their solution, see e.g. [3] and references therein, and [13]. Various linear algebra options have also been included in MEXX, depending on the substructure or sparsity of the matrices in (3.1). We proceed by describing some options which have been incorporated into MEXX (or will possibly be included in the near future.)

3.1 Direct matrix algorithms

- The zero option is to use the general LU–decomposition/substitution routines DGEFA/DGESL from LINPACK [10].
- In order to exploit the symmetry of (3.1) we offer the symmetric indefinite decomposition/substitution routines DSIFA/DSISL from LINPACK.
- When M is positive definite, one may compute the Choleski decomposition $M = LL^T$ and the QR–decomposition $L^{-1}G^T = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ to obtain the decomposition

$$\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} = \begin{pmatrix} L & 0 \\ GL^{-T} & R^T \end{pmatrix} \begin{pmatrix} I_{n_v} & 0 \\ 0 & -I_{n_\lambda} \end{pmatrix} \begin{pmatrix} L^T & L^{-1}G^T \\ 0 & R \end{pmatrix}. \tag{3.2}$$

  Solving (3.1) in this way corresponds to an algorithm analysed in [26]. It appears well–suited for mechanical formulations with full matrices, with a few (e.g. loop–closing) constraints, see the beginning of Section 1.3.

- Linear equations of the special form

$$\begin{pmatrix} M & 0 & K^T \\ 0 & 0 & J^T \\ K & J & 0 \end{pmatrix} \begin{pmatrix} \dot{v} \\ \ddot{q} \\ \mu \end{pmatrix} = \begin{pmatrix} f \\ 0 \\ h \end{pmatrix} \tag{3.3}$$

  with positive definite M, invertible K, and J of full column rank arise when one uses a minimal coordinate formulation without prior elimination of absolute coordinates, see Section 1.3. Their elimination and the reduction to minimal form can be mimicked in the solution of (3.3). Block Gaussian elimination gives a system of the form of the ODE that describes the motion in minimal coordinates:

$$\begin{aligned} A\ddot{q} &= b\,, \\ \text{with} \quad A &= J^T K^{-T} M K^{-1} J \\ b &= -J^T K^{-T}(f - MK^{-1}h)\,. \end{aligned} \tag{3.4}$$

  This system is solved for $\ddot{q}$. Then $\dot{v}$ is obtained from the last line of (3.3), and $K^T\mu$ is obtained from the first line. In mechanical terms, the symmetric positive definite matrix A represents a generalized mass matrix, and the first term of the expression defining b represents the generalized gyroscopic forces. While M, K, and J are usually sparse, A is in general a full matrix, because sparsity is lost in K^{-1}. Unless there are only few degrees of freedom in the mechanical system (whose number equals the dimension of A), the above reduction to minimal dimension is not recommended.

- To account for sparsity in (3.1), we have an option in MEXX which uses the public domain general sparse system solver MA28 from the Harwell library [12]. A more suitable choice would be the recent code MA47 of [13], which is tailored to sparse systems of the very form (3.1) satisfying (1.3). This code, however, is not in the public domain.

- When the Jacobian F of dry friction forces has to be taken into account, then symmetry in the matrix is lost, see (1.4b), (2.2). For this case we only provide DGEFA/DGESL of LINPACK for full matrices, and MA28 as a sparse solver.

3.2 Recursive elimination algorithms

Mechanical engineers have developed algorithms which solve the system of equations (3.1) at a computational cost which grows only linearly with the number of bodies in the system, at least in the case of tree–configured systems

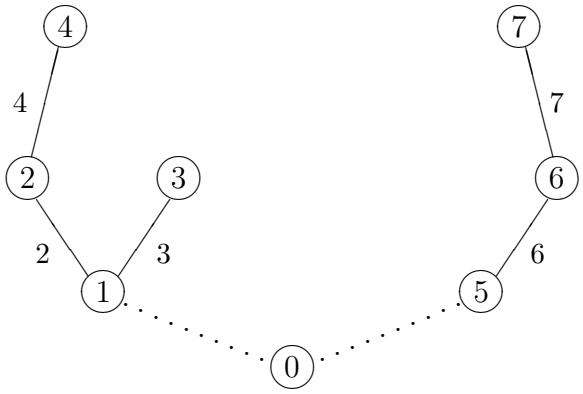


Figure 3.1: Tree–configured system

or systems with just a few closed kinematic loops, see e.g. [2, 4, 32, 36] and references therein. These algorithms, which are applicable to both absolute and absolute/relative coordinate formulations (see Section 1), can be viewed as body–oriented block Gaussian elimination on (3.1). We consider first the case of a multibody system with *kinematic tree structure:* Consider the graph whose nodes correspond to the bodies, and whose edges represent the joints in the system. When there are no closed loops, then the graph consists of *trees*. In each tree one body is singled out as the *root*. Every other body then has a unique *father* in the tree, which is its neighboring body on the path to the root. We formally introduce a node "0" as father of the roots. We assume that the tree is given a *monotone labelling*, that is, bodies are numbered from 1 to n_B such that the number of a body is always greater than that of its father. Joints are numbered such that a joint connecting a father and a son has the number of the son. See Figure 3.1, where e.g. $2 =$ father(4), and these two bodies are connected by joint 4.

In the linear system (3.1), we split the unknowns $w = (w_k)_{k=1}^{n_B}$ and $\lambda = (\lambda_k)$, where the subvector w_k corresponds to body k, and λ_k to joint k.

Formulation of joints by constraints: In a kinematic formulation as described in Section 1.2, the constraint equations in (3.1) of a joint connecting bodies k and $j = \text{father}\,(k)$ are of the form

$$G_k w_k + C_k w_j = g_k \; .$$

Here the dimension of w_k is 3 or 6 for planar or spatial mechanical systems, respectively. The mass matrix M is block–diagonal with 3×3 or 6×6 blocks M_k which are symmetric positive definite. G_k is assumed to have full row rank.

Consider now the equations of a *terminal body* k in the tree (one which has no sons):

$$\begin{aligned} M_k w_k + G_k^T \lambda_k &= f_k \\ G_k w_k &= g_k - C_k w_j \ . \end{aligned} \tag{3.5}$$

We can eliminate λ_k in this equation:

$$\lambda_k = -(G_k M_k^{-1} G_k^T)^{-1}(g_k - C_k w_j - G_k M_k^{-1} f_k) \ . \tag{3.6}$$

Now consider the equation for body j:

$$\begin{aligned} M_j w_j + G_j^T \lambda_j + \textstyle\sum_{k:j=\text{father}\,(k)} C_k^T \lambda_k &= f_j \\ G_j w_j &= g_j - C_j w_i \ . \end{aligned} \tag{3.7}$$

where $i = \text{father}\ (j)$. Inserting (3.6), this becomes

$$\begin{aligned} \hat{M}_j w_j + G_j^T \lambda_j &= \hat{f}_j \\ G_j w_j &= g_j - C_j w_i \end{aligned} \tag{3.8}$$

where

$$\begin{aligned} \hat{M}_j &= M_j + \textstyle\sum_{k:j=\text{father}\,(k)} C_k^T (G_k M_k^{-1} G_k^T)^{-1} C_k \\ \hat{f}_j &= f_j + \textstyle\sum_{k:j=\text{father}\,(k)} C_k^T (G_k M_k^{-1} G_k^T)^{-1}(g_k - G_k M_k^{-1} f_k) \ . \end{aligned} \tag{3.9}$$

We can now eliminate λ_j, and so on, climbing down the tree recursively. When method (3.2) is used to solve the subsystems (3.8), this leads to the following algorithm:

Decomposition: for $k = n_B, n_B - 1, \ldots, 1$ do

$$\begin{aligned} M_k &= L_k L_k^T && \text{Choleski decomposition} \\ G_k^T &:= L_k^{-1} G_k^T \\ G_k^T &= Q \begin{pmatrix} R_k \\ 0 \end{pmatrix} && \text{QR–decomposition} \\ C_k &:= R_k^{-T} C_k \\ M_j &:= M_j + C_k^T C_k && \text{with } j = \text{father}\,(k) \ (\ \text{omit if } j = 0) \end{aligned} \tag{3.10a}$$

Forward substitution: for $k = n_B, n_B - 1, \ldots, 1$ do

$$\begin{aligned} f_k &:= L_k^{-1} f_k \\ g_k &:= R_k^{-T}(G_k f_k - g_k) \\ f_j &:= f_j - C_k^T g_k && \text{with } j = \text{father}\,(k) \ (\text{omit if } j = 0) \end{aligned} \tag{3.10b}$$

Back substitution: for $k = 1, \ldots, n_B$ do

$$\begin{aligned} g_k &:= g_k + C_k w_j \qquad \text{with } j = \text{father}\,(k) \text{ (omit if } j = 0) \\ \lambda_k &:= R_k^{-1} g_k \\ w_k &:= L_k^{-T}(f_k - G_k^T \lambda_k)\ . \end{aligned} \tag{3.10c}$$

Formulation using joint coordinates: In the kinematic formulation of Section 1.3, upon splitting $w_k = \begin{pmatrix} y_k \\ z_k \end{pmatrix}$ in components $y_k \in \mathbb{R}^6$ (or $\mathbb{R}^3$) and $z_k \in \mathbb{R}^{d_k}$ (d_k = number of degrees of freedom of joint k) which contain body and joint variables, respectively, the equations (3.1) take the following form:

$$\begin{pmatrix} M_k & 0 & K_k^T \\ 0 & 0 & J_k^T \\ K_k & J_k & 0 \end{pmatrix} \begin{pmatrix} y_k \\ z_k \\ \mu_k \end{pmatrix} = \begin{pmatrix} f_k \\ 0 \\ h_k - H_k y_j \end{pmatrix} - \sum_{l:k=\text{father}\,(l)} \begin{pmatrix} H_l^T \mu_l \\ 0 \\ 0 \end{pmatrix} \tag{3.11}$$

with $j = \text{father}\,(k)$. Here M_k, K_k, H_k are 6×6 (or 3×3) matrices, with M_k symmetric positive definite and K_k invertible. The $6 \times d_k$ matrix J_k has full column rank.

For a terminal body k, the last sum in (3.11) does not appear, and the corresponding variables can be eliminated using the reduction procedure (3.4). With the notation

$$\bar{J}_k = K_k^{-1} J_k,\ \bar{H}_k = K_k^{-1} H_k,\ \bar{h}_k = K_k^{-1} h_k\ , \tag{3.12}$$

and (cf. (3.4))

$$A_k = \bar{J}_k^T M_k \bar{J}_k\ , \tag{3.13}$$

this yields

$$\begin{aligned} z_k &= -A_k^{-1} \bar{J}_k^T (f_k - M_k \bar{h}_k + M_k \bar{H}_k y_j) \\ y_k &= \bar{h}_k - \bar{H}_k y_j - \bar{J}_k z_k \\ K_k^T \mu_k &= (I - M_k \bar{J}_k A_k^{-1} \bar{J}_k^T)(f_k - M_k \bar{h}_k + M_k \bar{H}_k y_j)\ . \end{aligned} \tag{3.14}$$

Substituting μ_k into the equations (3.11) of father j thus gives

$$\begin{pmatrix} \hat{M}_j & 0 & K_j^T \\ 0 & 0 & J_j^T \\ K_j & J_j & 0 \end{pmatrix} \begin{pmatrix} y_j \\ z_j \\ \mu_j \end{pmatrix} = \begin{pmatrix} \hat{f}_j \\ 0 \\ h_j - H_j y_i \end{pmatrix} \tag{3.15}$$

with $i = \text{father}\,(j)$, and

$$\begin{aligned} \hat{M}_j &= M_j + \sum_{k:j=\text{father}(k)} \bar{H}_k^T (M_k - M_k \bar{J}_k A_k^{-1} \bar{J}_k^T M_k) \bar{H}_k \\ \hat{f}_j &= f_j - \sum_{k:j=\text{father}(k)} \bar{H}_k^T (I - M_k \bar{J}_k A_k^{-1} \bar{J}_k^T)(f_k - M_k \bar{h}_k) \; . \end{aligned} \tag{3.16}$$

We note that $\hat{M}_j$ is again symmetric positive definite. We can now eliminate the variables of body j, and so on recursively down to the root of the tree. When one assumes that the last equation in (3.11) has been premultiplied by K_k^{-1} a priori and μ_k replaced by $K_k^T \mu_k$ (so that (3.11) now holds with $K_k = I$ for all k), then this gives the following algorithm:

Decomposition: for $k = n_B, n_B - 1, \ldots, 1$ do

$$\begin{aligned} \begin{pmatrix} A_k & B_k^T \\ B_k & C_k \end{pmatrix} &:= \begin{pmatrix} J_k^T \\ H_k^T \end{pmatrix} M_k (J_k, H_k) \\ A_k &= L_k L_k^T && \text{Choleski decomposition} \\ (J_k^T, B_k^T) &:= L_k^{-1} (J_k^T, B_k^T) \\ M_j &:= M_j + C_k - B_k B_k^T && \text{with } j = \text{father}\,(k) \text{ (omit if } j = 0) \; . \end{aligned} \tag{3.17a}$$

Forward substitution: for $k = n_B, n_B - 1, \ldots, 1$ do

$$\begin{aligned} f_k &:= f_k - M_k h_k \\ g_k &:= J_k^T f_k \\ f_j &:= f_j - H_k^T f_k + B_k g_k \quad \text{with } j = \text{father}\,(k) \text{ (omit if } j = 0) \; . \end{aligned} \tag{3.17b}$$

Back substitution: for $k = 1, \ldots, n_B$ do

$$\begin{aligned} g_k &:= g_k + B_k^T y_j \qquad \text{with } j = \text{father}\,(k) \text{ (omit if } j = 0) \\ z_k &:= -L_k^{-T} g_k \\ y_k &:= J_k g_k - H_k y_j \\ \mu_k &:= f_k - M_k y_k \\ y_k &:= y_k + h_k \; . \end{aligned} \tag{3.17c}$$

In a multibody system with few closed kinematic loops, one may first formulate the equations for a subtree obtained by cutting loops, and then add the loop–closing constraints. This leads to a system matrix of the form

$$\begin{pmatrix} T & C^T \\ C & 0 \end{pmatrix}$$

where T is the system matrix of the tree configured system. Block Gaussian elimination then leads to a linear system with matrix $CT^{-1}C^T$ of the dimension of the loop–closing constraints. Here the product $T^{-1}C^T$ is formed by applying the recursive elimination algorithm to the columns of C^T.

3.3 Preconditioned iterative solvers

In the integration routine, a sequence of linear systems (3.1) has to be solved, where the system matrices differ only slightly between successive iteration steps. This suggests the use of iterative techniques preconditioned by using a matrix decomposition which is redone only at the beginning of a basic (integration and extrapolation) step, but not in the intermediate steps.

Formulation of joints by constraints: A system (3.1) with *positive definite* matrix M can be solved by first computing λ from

$$GM^{-1}G^T \cdot \lambda = GM^{-1}f - g \tag{3.18a}$$

and then determining w from

$$w = M^{-1}f - G^T\lambda \; . \tag{3.18b}$$

When, as usual, M is block–diagonal (and constant), then the interest is in the iterative solution of (3.18a). An efficient obvious choice is to use conjugate gradients preconditioned by a neighboring matrix $G_0M_0^{-1}G_0^T$. The algorithm requires to form matrix vector products $GM^{-1}G^T\lambda$ (without explicitly forming $GM^{-1}G^T$). Also $G_0M_0^{-1}G_0^T$ is not formed and decomposed explicitly, since the solution of $G_0M_0^{-1}G_0^T\lambda = r$ can be obtained from a system of the form (3.1),

$$\begin{pmatrix} M_0 & G_0 \\ G_0^T & 0 \end{pmatrix} \begin{pmatrix} w \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ -r \end{pmatrix} \; . \tag{3.19}$$

This is solved by using a decomposition from Section 3.1 or 3.2.

Formulation using joint coordinates: A similar approach is also feasible when the system matrix is of the form (3.3). Here (3.4) is solved using preconditioned conjugate gradients. The matrix A need not be formed explicitly, and the solution of $A_0z = r$ with $A_0 = (K_0^{-1}J_0)^T M_0 (K_0^{-1}J_0)$ is obtained from

$$\begin{pmatrix} M_0 & 0 & K_0^T \\ 0 & 0 & J_0^T \\ K_0 & J_0 & 0 \end{pmatrix} \begin{pmatrix} y \\ z \\ \mu \end{pmatrix} = \begin{pmatrix} 0 \\ r \\ 0 \end{pmatrix} \; , \tag{3.20}$$

which is solved by a straightforward modification of algorithm (3.17). At present, iterative procedures are not available within MEXX.

Chapter 4

Implementation

This section gives a short introduction in how to use the code MEXX. A detailed documentation including all technical details is part of the code and not reproduced here. Rather, the relation of the software to the underlying algorithms is pointed out and some general remarks are made. The code is written in ANSI standard FORTRAN77 (double precision) and belongs to the numerical software library CodeLib of the Konrad–Zuse–Zentrum Berlin, thus it is available for interested users.

Except for the special linear algebra methods of Section 3.2 and 3.3, all algorithms described in the preceding sections are realized in only one piece of numerical software – the package MEXX (Revision 1.1). The whole package consists of a set of subroutines, where the user interface subroutine MEXX must be called from a user written driver program. All communication between the package and the user is done via the arguments of the calling sequences of MEXX and four user supplied subroutines, respectively. There are calls from MEXX to a subroutine, say FPROB, which has to supply the problem describing functions and matrices T, M, f, G, g, g^I, d, C – cf.(1.1,2.11). Strictly speaking, instead of providing the matrix $T(t,p)$ the routine FPROB must return the matrix vector product $\dot{p} = T(t,p) \cdot v(t)$. If the modified discretization (2.2a) is used, FPROB must provide the Jacobian matrix $F := \partial f / \partial \lambda$ additionally. There may be calls to a subroutine, say FSWIT, which has to return the values of the switching functions ϕ_i – cf. (2.16).

Finally, the current solution at the internally selected integration points is passed to a subroutine, say SOLOUT, whereas the current solution at (eventually) prescribed dense output points is passed to another subroutine, say DENOUT. In order to have the software as flexible as possible all four subroutines are input arguments for MEXX. A dummy routine FSWIT and example routines SOLOUT and DENOUT are part of the package.

Besides the communication mentioned above, MEXX may write special output to some FORTRAN units. For example, MEXX offers the feasibility to dump the continuous representation of the solution. With that, an easy and efficient way of postprocessing the solution, e.g. for graphical purposes, is given.

At the end, we would like to mention a special characteristic of the software. The code which performs the numerical integration and the code which performs the matrix manipulations, e.g. linear system solution and matrix vector

products, is separated as far as technically possible. Therefore, the package is open for an adaptation or extension of the currently implemented linear algebra software.

An overview on the current program structure is given in the appendix.

4.1 Interfaces

The calling sequence of the user interface routine MEXX reads:

```
SUBROUTINE MEXX (NP, NV, NL, NG, NU,
                 FPROB, T, TFIN, P, V, U, A, RLAM,
                 ITOL, RTOL, ATOL,
                 H, MXJOB, IERR, LIWK, IWK, LRWK, RWK,
                 SOLOUT, DENOUT,
                 NSWIT, FSWIT, ISWIT)
```

Within this calling sequence one may distinguish two groups of arguments: one group for the problem definition and another which refers to the algorithm and its implementation. The arguments of the first group are:

`NP,NV,NL,NG,NU` – integer, input

the dimensions $n_p, n_v, n_\lambda, n_g, n_u$

`P,V,U,A,RLAM` – real arrays, in–out:

in: the initial values $p(t_0), v(t_0), u(t_0), (a(t_0), \lambda(t_0))$

out: the current values $p(t), v(t), u(t), a(t), \lambda(t)$

`T,TFIN` – real, in-out/in:

in: the starting(t_0)/final(t_{end}) point of integration

out: the current point of integration

`FPROB` – external subroutine, input:

evaluation of the problem functions $\dot{p}, M, f, G, g, g^I, d, C$

`FSWIT` – external subroutine, input:

evaluation of the switching functions $\phi_i, i = 1, ..., s$

`NSWIT` – integer, input:

number of switching functions (the dimension s in (2.16))

`ISWIT` – integer array, output

indicates for which switching function the zero was found

`SOLOUT` – external subroutine, input:

output routine, called at integration points

`DENOUT` – external subroutine, input:

output routine, called at intermediate dense output points

`ITOL,RTOL,ATOL` – integer, real arrays/scalars, input:

required accuracy of the solution

In order to make a proper setting of the above arguments, recall that for the discretization (2.1) $a(t_0)$ is not required and $\lambda(t_0)$ only, if the functions f or d depend non–linearly on λ. However, if dense output is required, these vectors should contain consistent initial values or the corresponding option for an internal approximation should be switched on (MXJOB(22)=1).

The prescribed values for RTOL and ATOL enter via (2.8a,b) into the internal error estimate (2.7). The integer flag ITOL is used to indicate whether these values are scalars (ITOL=0) or arrays (ITOL=1) of dimension $n_p + n_v + n_u$. Further details can be found in the code documentation.

For a precise description of the arguments of the external subroutines FPROB, FSWIT, SOLOUT, DENOUT we refer again to the documentation in the code. However, some features of the function evaluation are worth mentioning. Self–evident, FPROB must provide the function values for that arguments t, p, v, λ, u which are input to FPROB. But, observe that constant functions have to be set within each call again.

Due to the special discretization and the additional projection step, not all functions must be provided simultaneously at each call of FPROB. Rather, one can distinguish 7 different types of calls. First, for given input arguments t, p, v, λ, u the functions $\dot{p}, f$ must be provided. Second, for given (new) values t, p the functions M, G, g^I must be provided. Third, for given input arguments t, p, v, λ, u the function d has to be computed. These types of calls are needed to perform the discretization steps (2.1). The second type of call is also used in the projection step for the velocity (2.12). Furthermore, for varying p the functions M, C, g – cf. (2.10,2.11a) are required, for varying p the function g (for (2.11a)) and for fixed p but varying v the function $\dot{p} = T(t, p) \cdot v(t)$ (for (2.11b)). Finally, if the modified discretization (2.2a) is used, then for given t, p, v, λ, u the functions $\dot{p}, f$ and, in addition, the Jacobian F have to be computed. This type of call replaces some of the calls of the first type. The information which function(s) has(have) to be evaluated at a specific call of FPROB is passed to FPROB via its argument list (an array of logical flags).

An alternative implementation would have been to separate totally the different function calls. However, if the functions share common terms (expensive to evaluate), a multiple recomputing can easily be avoided by exploiting the special design of the function evaluation within MEXX.

The second group of arguments is:

`H` – real, in–out:

in: the initial stepsize

out: the current stepsize guess

`MXJOB` – integer array, in–out:

in: the first 50 elements may be used to specify special options

out: the elements 51 up to 150 contain performance information

`IERR` – integer, output:

error indicator

`LIWK,LRWK` – integer, input:

declared lenghts of integer and real workspace arrays

`IWK,RWK` – integer/real arrays, in–out

workspace arrays, partially used for further optional input and output.

Besides providing workspace (arrays IWK, RWK), all these arguments may be used to control, modify and monitor the performance of MEXX. This can be done by assigning special values to (part of) the first elements of MXJOB, IWK and RWK. Note that a zero initiation forces an internal assignment with default values. On output, some of the elements will hold helpful information, e.g. the minimal required lengths of IWK and RWK.

4.2 Options

Discretization and linear algebra mode

The most essential variants and modifications which can be selected by the user of MEXX are certainly the type of the basic discretization and the linear algebra mode. For the standard discretization (2.1) the most appropriate mode for the linear system solution may be chosen according to the problem formulation in hand – cf. Section 3.1. If no specification is made by the user, a general full mode solution is done in order to solve the systems of type (3.1). We ignore the symmetry of the system, as our numerical experiments revealed that the general decomposition/solve routines DGEFA/DGESL from LINPACK are mostly faster than their symmetric counterparts DSIFA/DSISL – cf. Section 5.

Using the modified discretization (2.2), however, the choice is anyway restricted as the matrix to be decomposed is no longer symmetric. Here, only the full or sparse mode LU–decomposition/substitution may be selected.

The following Table 4.1 shows how to choose the discretization and the linear algebra mode. We allow also to prescribe the maximum order k_{max} of the method as this may be a helpful tool for problems with a restricted order of differentiability.

Option	Selection	Value	Consequence
Discretization	MXJOB(1)	0	standard discretization
		1	modified discretization
solution mode for linear systems	MXJOB(2)	0	full mode LU
		1	full mode UDU^T
		2	full mode $LL^T + QR$
		3	special block Gaussian elimination
		10	sparse mode LU
storage mode for matrix M	MXJOB(3)	0	full mode
		1	block diagonal
storage mode for matrix G (and F)	MXJOB(6)	0	full mode
		1	sparse mode
k_{max}	MXJOB(9)	3 . . . 13	order restriction$\leq k_{max}$

Table 4.1: Options for the main variants of MEXX

Sparse Linear System Solution

If the general sparse linear algebra mode is switched on, the user has to provide the matrix G (and the matrix $F = \frac{\partial f}{\partial \lambda}$ if the modified discretization is used) in sparse form, i.e. besides the numerical values (only the nonzero elements) the pattern of the matrix (matrices) must be provided additionally. The mass matrix M should be given in a special storage mode which exploits the block diagonal structure of this matrix. As pointed out above the sparse linear system solution is done by means of the MA28 package from Harwell. The compilation of MEXX and MA28 follows the lines presented in [11] . Thus we give only a short summary.

Within the MA28 package there are two routines to factor a given matrix. The expensive Analyse/Factorize routine MA28A analyses the matrix and tries to minimize the number of fill–in elements in its LU–decomposition. Besides, there is the fast Factorize routine MA28B which factors a matrix with the same nonzero pattern as from a previous call to MA28A. But the values may have changed, thus the now prescribed pivot sequence may become not appropriate. This is internally checked and one may restart with a factorization by MA28A, if numerical instability is indicated. Provided that the structural nonzero pattern of the matrix to be decomposed is known exactly and does not change within the course of the integration, after a first decomposition with MA28A, the fast factor routine may be used, in principle, throughout the whole integration. Numerical experience shows, that during an integration quite often just 1–3 expensive Analyse/Factorize calls are

necessary. All other decompositions can be done by the fast FACTORIZE routine.

But this technique can be applied in MEXX only if the standard discretization is used and the matrix required for the projection step of the position vector, cf. (2.11a), is of the same type as for the discretization steps, cf. (2.1b). Otherwise, the matrix must be factored by the ANALYSE/FACTORIZE routine MA28A whenever a new type arises. However, even in the worst case (modified discretization and $C \neq G$) just 3 calls of MA28A are required per basic step – one at the very beginning of the discretization steps, one for the projection steps (2.11a) and one for the projection step (2.12). As the dominating part of the decompositions is required to perform the discretization, where one can mostly apply the fast FACTORIZE routine, the sparse linear system solution works still quite efficiently.

Monitor Output

In critical applications it is often helpful to monitor the performance of a code by studying characterizing quantities. Based on this information, optional variants and modifications of the basic method can be selected. The amount of monitor output produced by MEXX is controlled by some output flags and directed to associated FORTRAN units. In order to monitor the algorithmic performance of the code, the internal flag INTMON can be modified by setting the associated element of the MXJOB array (INTMON corresponds to MXJOB(13) and the associated output unit LUINT to MXJOB(14)). An user assignment of INTMON=0 produces no monitor output, whereas a setting INTMON=3 will generate a detailed integration monitor, e.g. the current integration point, stepsize and order as well as the error estimates of the extrapolation tableau and of the projection step. Similar flag/unit pairs are available for error messages, general printout, and time monitor printout – see Table 4.2.

Solution and Dense Output

Recall that two output routines, SOLOUT and DENOUT, must be passed to MEXX. These routines may be user written routines or the example routines added to the package. But, as long as the standard options are in use, these routines are not called by MEXX.

Specifying MXJOB(30)=1 entails that SOLOUT is called, first of all at t_0, the starting point of integration, and then after each successful integration step. To be precise, the extrapolated and projected solution approximations $p(t), v(t)$ and the extrapolated values $u(t), a(t), \lambda(t)$ are passed to SOLOUT. As projection is done also at t_0, the projected values $p(t_0), v(t_0)$ must not

necessarily coincide with the user given starting values.

Option	Selection	Range	Default	Unit
error/general printout	MXJOB(11)	0–2	0	MXJOB(12)
integration monitor	MXJOB(13)	0–5	0	MXJOB(14)
contin. solution dump	MXJOB(17)	0–1	0	MXJOB(18)
time monitor	MXJOB(19)	0–1	0	MXJOB(20)
type of SOLOUT calls	MXJOB(30)	0–1	0	—
type of DENOUT calls	MXJOB(31)	0–4	0	—
No. of interpol. points	MXJOB(32)	≥ 0	0	—
component selection	MXJOB(33)	0–1	0	—

Table 4.2: Options for output generation

The dense output routine DENOUT is called only if the associated flag (MDOUT=MXJOB(31) is set to a positive value. Within the current implementation of MEXX, four dense output modes are available.

MDOUT = 1 Interpolation at NDOUT points which are uniformly distributed between t_0 and t_{end}. Additionally t_0, t_{end} are considered. DENOUT is called NDOUT+2 times

MDOUT = 2 Interpolation at NDOUT points within each integration interval $[t, t + H]$. Additionally all integration points are considered. DENOUT is called NSTEP*(NDOUT+1) + 1 times.

MDOUT = 3 Interpolation such that the maximum interval between two successive interpolation points is less than or equal to a value Δt^{IP}_{max}. This means, if the current integration interval is greater than Δt^{IP}_{max} it is uniformly subdivided such that the required condition holds. Additionally all integration points are considered. DENOUT is called at least at all integration points.

MDOUT = 4 Interpolation at NDOUT user prescribed output points $t^{IP}_i, i = 1, ..., ndout$. All integration points may be ignored. DENOUT is called exactly NDOUT times.

The required number NDOUT can be prescribed via MXJOB(32). The requested output points are input to MEXX via RWK(51), ..., RWK(50+NDOUT). The latter values must be in increasing order, i.e. $t^{IP}_i < t^{IP}_{i+1}$. For MDOUT = 3, Δt^{IP}_{max} must be set via RWK(51).

In general, the additional amount of work to perform the continuous output within an extrapolation method is comparatively small. In special situations, however, things may change. Thus, MEXX offers the chance to select only part of the the vectors p, v, u, a, λ for the dense output generation. This information is passed to MEXX by prescribing the selected components in special positions of IWK. This selection can additionally be switched on/off by setting MXJOB(33)=1/0. If the continuous output is only requested for graphical purposes, there is another way of reducing computing time – as well as storage. The user may prescribe a maximum order (via MXJOB(34)) for the interpolating polynomial.

Finally, the internal representation of the continuous output, i.e. the coefficients of the interpolating Hermite polynomial, can be dumped to a special FORTRAN unit. This option is switched on by setting MXJOB(17)=1 (associated unit: MXJOB(18)). This allows an easy, cheap and flexible postprocessing. An example program, which realizes such a postprocessing is added to the MEXX package. Note that this option only works if the simultaneous dense output option (MXJOB(31)) is switched on also. Furthermore, only the data for the thereby selected (or all) components are dumped.

Root finding

In the standard case, the event location option of MEXX is switched off. Thus, the input arguments NSWIT, ISWIT and FSWIT are ignored and may be dummy.

The root finding algorithm is switched on by setting MXJOB(35) to a valid positive value. The choice MXJOB(35)=1 forces MEXX to return to the calling program after having located the first (nearest to t_0) root in one of the switching functions ϕ_i – cf. (2.16).

The choice MXJOB(35)=2 allows MEXX to continue the integration. In both cases, the output routine SOLOUT is called as if the root points would be integration points. Except for these additional calls of the output routine SOLOUT, the performance of MEXX solving the problem up to t_{end} is not changed – compared to an integration without root finding.

Recall that for the current integration interval $[t, t + H]$ the root finding algorithm is activated only, if the sign of at least one switching function ϕ_i has changed, i.e.

$$\phi_i(t) \cdot \phi_i(t + H) < 0 \tag{4.1a}$$

holds. But in this form the check is not suited for implementation. Rather, this check is performed only, if the condition

$$\phi_i(t) \geq \phi_{res} \wedge \phi_i(t + H) \geq \phi_{res},\ \phi_{res} \geq 0 \tag{4.1b}$$

holds additionally. With that, a switching function ϕ_i where $\phi_i \approx 0$ holds for a certain period of time will cause no trouble. The maximal allowed residual ϕ_{res} may be specified by the user.

Locating a zero requires the continuous representation of all components of the vectors p, v, u, a, λ. First, as ϕ_i is allowed to depend on all these unknowns. Second, as the output of these values at the root point(s) or the return to the calling program, respectively, requires the computation of solution values for all the unknowns. But this representation is only necessary for such integration intervals where the check for a sign change indicates the occurrence of a root. For all other intervals, in order to avoid computational overhead, the dense output values are determined only for the user selected components. In any case, only the user selected components are passed to DENOUT, even if the other values are available due to a zero location.

In the very special situation where a switching function possesses an even number of zeros or a zero with even multiplicity in one integration interval $[t, t+H]$ – a case typical for constructed test problems – the indicator (4.1a/b) is not appropriate. Thus, we offer the option to perform the sign check not only at the endpoint of the current integration interval but at some checking points, say $t_{c_l}, t_{c_l} := t + l\frac{H}{n_c},\ l = 0, 1, \ldots, n_c$, by

$$\phi_i(t_{c_{l-1}}) \cdot \phi_i(t_{c_l}) < 0$$

combined with a condition of type (4.1b). The number of checking points n_c may be prescribed by the user. Note that a choice $n_c > 1$ requires the computation of the dense output representation throughout the whole integration.

Time Monitor

In order to get detailed information concerning the performance of MEXX a time monitor package, which is designed for time measurements of different, possibly nested program parts, is added to MEXX. It may be used to get the information which parts of the algorithm are the most time consuming ones. The monitor is turned on by setting IOPT(19)=1. Its output will be written to the FORTRAN unit IOPT(20).

Because of machine dependence the user has to adapt the subroutine ZIBSEC in such a way that on output the only argument of this routine contains a "time stamp", measured in seconds. As distributed, ZIBSEC is a dummy routine which always returns zero to this argument.

Chapter 5

Numerical Experiments

In this section we present some numerical results illustrating the performance of MEXX. All experiments have been carried out on a SUN SPARC1+ Workstation using the Sun FORTRAN Compiler Version 1.4.1 with standard options except for the optimization which had to be switched off due to buggy CPU timing for optimized code. Unless stated otherwise, the standard options and parameters of MEXX are active, i.e. no root finding, no dense output and full mode linear algebra, neglecting any symmetry. For all problems we have chosen an initial stepsize of $H_{init} = 10^{-3}$.

It is not the purpose of this section to make a comparison to other codes, such as ODASSL by Führer, DAESOL by Eich, or HEM by Brasey and Hairer. Rather, we would like to compare some of the algorithmic variants available in the software package MEXX.

Sensible measuring units for MEXX are the number of linear system solutions and the number of problem function evaluations. Recall that a linear system solution mostly means a matrix decomposition followed by a forward/backward substitution. Only in the projection step the matrix is fixed and one decomposition may be followed by more than one substitution – if more than one Newton iteration is required.

Concerning the counting of problem functions, recall that the number of calls for the various functions $(M, f, G, g, ..)$ differ from each other. But, counting those number calls of the problem routine FPROB where the simultaneous evaluation of M, G, g^I is requested, one has a quite good indicator. The number of FPROB evaluations where the user has to provide $f, \dot{p}$ or d is slightly less, but directly coupled with the (M, G, g^I) count.

All other function evaluations, e.g. those needed to perform the projection step, can be neglected in most cases.

5.1 Cable Drum with Dry Friction

The equations of this simple test problem model a rotating cable drum with connected load, cf. [30], p. 121. The model includes dry friction and a linear damping of the load. The whole configuration is depicted in Figure 5.1. Herein, y_1, m_1 denote the height and mass of the load and $x_2, y_2, \alpha_2, m_2, I_2$ the position, mass and inertia of the cable drum. With that the constrained equations of motion to be integrated read

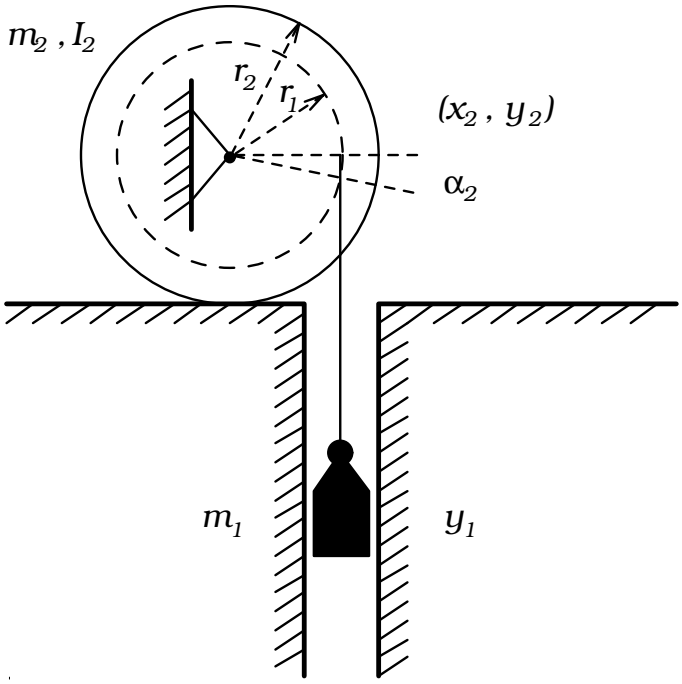


Figure 5.1: Cable drum with dry friction

$$\begin{array}{rcl} \dot{p} & = & v \\ M(t,p)\cdot\dot{v} & = & f(t,p,v,\lambda) - G(t,p)^T\lambda \\ 0 & = & G(t,p)\cdot v + g^I(t,p) \\ 0 & = & g(t,p) \end{array} \tag{5.1}$$

where

$$\begin{array}{rcl} p & = & (y_1, x_2, y_2, \alpha_2)^T\ , \quad v = (\dot{y}_1, \dot{x}_2, \dot{y}_2, \dot{\alpha}_2)^T \\ M & = & diag(m_1, m_2, m_2, I_2) \\ f & = & (-m_1 g_{grav} - c_{damp}\dot{y}_1,\ -\mu\lambda_2,\ -m_2 g_{grav}, -\mu r_2\lambda_2)^T \\ g & = & (x_2,\ y_2 - r_2,\ y_1 - y_2 - \alpha_2 r_1)^T \\ g^I & = & \partial g/\partial t = 0 \\ G & = & \partial g/\partial p\ . \end{array} \tag{5.2}$$

For the problem parameters we use the values

$$\begin{array}{l} m_1 = 10\ ,\ \ m_2 = 1\ ,\ \ I_2 = 1\ ,\ \ r_1 = 1\ ,\ \ r_2 = 1 \\ g_{grav} = 1\ ,\ \ c_{damp} = 1\ ,\ \ \mu = 0.25\ . \end{array} \tag{5.3}$$

Consistent initial conditions are given by

$$\begin{aligned}
p_1 &\quad \textit{to be chosen} \ \ (\leq 1) \\
p_2 &= 0\ , \ \ p_3 = r_2\ , \ \ p_4 = (p_1 - r_2)/r_1 \\
v_1 &\quad \textit{to be chosen} \\
v_2 &= 0\ , \ \ v_3 = 0\ , \ \ v_4 = v_1/r_1 \\
\lambda_3 &= \frac{(-m_1 I_2 g_{grav} - c_{damp}\dot{y}_1 I_2 - m_1 M_2 r_1 r_2 g_{grav} \cdot \mu)}{I_2 + m_1 r_1^2 - m_1 r_1 r_2 \mu} \\
\lambda_2 &= \lambda_3 - m_2 g_{grav} \\
\lambda_1 &= -\mu\lambda_2 \\
a_1 &= (-g_{grav} m_1 - c_{damp}\dot{y}_1 + \lambda_3)/m_1 \\
a_2 &= 0\ , \ \ a_3 = 0\ , \ \ a_4 = a_1/r_1\ .
\end{aligned} \tag{5.4}$$

For the numerical experiments presented in this paper we take:

$$p_1 = 0\ , \ \ v_1 = 0, \tag{5.5}$$

and as simulation interval we choose $[t_0, t_{end}] = [0, 4]$. Within this time, the cable drum performs approximately one rotation if friction is absent.

The main difficulty for the numerical solution of this problem is the occurrence of dry friction. The question is, how long the usual discretization (2.1) can be used as the friction coefficient μ increases. Thus, we have to study the Jacobian $F := \partial f/\partial\lambda$. For the *cable drum* problem there are only two nonzero elements in the Jacobian, $F_{2,2} = -\mu, F_{4,2} = -\mu r_2$. According to the considerations made in Section 2, the basic discretization is only applicable as long as $\|F\| \ll 1$, i.e. as long as $\mu \ll 1$ ($as\ r_2 = 1$).

We illustrate the effect of increasing dry friction by the following experiment. The problem is solved for different values of μ with both discretizations, the standard (*std*) discretization (2.1) and the modified (*mod*) discretization (2.2). The tolerances are set to $RTOL = ATOL = 10^{-5}$. As the main indicators for the performance we report the CPU time (CPU), the number of integration steps (*nstep*), the number of linear system solutions (*nsol*) and the number of function (M, G, g^I) – evaluations (*nfcn*). The results are summarized in Table 5.1. The fail runs of standard MEXX for $\mu > 1$ are due to minimum permitted stepsize reached.

5.2 Seven Body Mechanism

The seven body mechanism or "Andrews squeezing mechanism" has become a popular benchmark problem. For a detailed problem description see [21, 31] and references therein. The equations, even part of the FORTRAN code, can be found in [21].

discr. mode	dry friction coefficient	*nstep*	*nfcn*	*nsol*	CPU
std	0.0	7	99	107	0.23
	0.125	8	102	111	0.23
	0.25	9	112	122	0.22
	0.5	10	162	173	0.34
	0.75	23	273	297	0.66
	1.0	76	1082	1159	2.37
	1.25	47	–	–	–
	1.5	23	–	–	–
mod	0.0	7	99	107	0.24
	0.125	7	99	107	0.20
	0.25	7	99	107	0.22
	0.5	7	99	107	0.20
	0.75	7	105	114	0.25
	1.0	7	87	95	0.25
	1.25	9	149	159	0.29
	1.5	8	114	123	0.27

Table 5.1: Performance of MEXX for different dry friction coefficients (standard versus modified discretization)

First, we report the results of solving this problem with MEXX for different required tolerances $RTOL = ATOL$. Recall that the standard option for the linear system solution is the general full mode LU–decomposition/substitution (DGEFA/DGESL of LINPACK). The surprising fact that the symmetric variant (DSIFA/DSISL from LINPACK) does not work faster, is now exemplified by presenting the CPU times of MEXX runs for both variants. We present the usual indicators (*nstep, nfcn, nsol*, CPU) for the standard version only, but the portion of CPU time, which is needed by the linear algebra routines is given for both variants – see Table 5.2.

The result given in the last row of this table is no contradiction to the above statement as, due to the different roundoff errors of the linear algebra methods, the performance of the integration is affected for the tolerances 10^{-9} and 10^{-11}. With the symmetric mode switched on, MEXX requires about 3% and 11% respectively less function evaluations and solves.

Concerning the general performance of MEXX, the results nicely show the typical behaviour of an extrapolation method. More stringent tolerance requirements result, mainly, in an increase of the order. So, the number of solves and function evaluations (again the (M, G, g^I) – evaluation) increases

Required Tolerance	*nstep*	*nfcn*	*nsol*	CPU(s)	lin. alg. CPU	
					LU	UDU^T
10^{-3}	17	488	511	3.2	2.1	2.3
10^{-5}	26	925	957	6.2	3.8	4.1
10^{-7}	23	1530	1563	10.3	6.7	7.4
10^{-9}	32	2533	2571	17.3	11.0	11.1
10^{-11}	37	3554	3611	25.5	15.9	14.2

Table 5.2: Results of Standard–MEXX solving the Standard–Problem

by a factor of 7 whereas the number of steps is just doubled (comparing 10^{-11} vs. 10^{-3}).

Now, as an additional experiment, we solve the problem with the dense output option switched on (for all components). As expected, the overall CPU time increases, but the additional amount of work (19 % for the run with $RTOL = ATOL = 10^{-5}$) is quite modest.

Finally, in order to illustrate the performance of the root finding algorithm, we switch on this option but allow MEXX to continue the integration up to $t_{end} = 0.03$. As a simple switching function we use

$$\ddot{\beta} = 0 \; .$$

Herein, β denotes the angle of that body which is connected to the motor of the mechanism. MEXX reports all five roots

$$t_{root} = 1.124 \; , \quad 1.602 \; , \quad 2.147 \; , \quad 2.462 \; , \quad 2.998$$

without any problems. Although the acceleration vector does not enter into the error control of MEXX, the maximum relative error of the root values turns out to be $3{\cdot}10^{-4}$. Compared to the above mentioned standard run (with $RTOL = ATOL = 10^{-5}$) the overall CPU time is only slightly increased (13%). This increase is less than the increase for the run with the dense output option switched on, as the dense output values are required in 5 (of 26) integration intervals only. The additional amount of work needed to perform the Newton iterations (3 steps to get the root of the unprojected solution, 1 step for the projected solution – on average) is less than the work to perform an overall dense output representation.

5.3 Nonlinear Truck Model

Recently a planar vertical truck model with nonlinear suspension elements for the interconnection of the axle and the chassis was proposed [33] – cf. Figure 5.2 for an illustration.

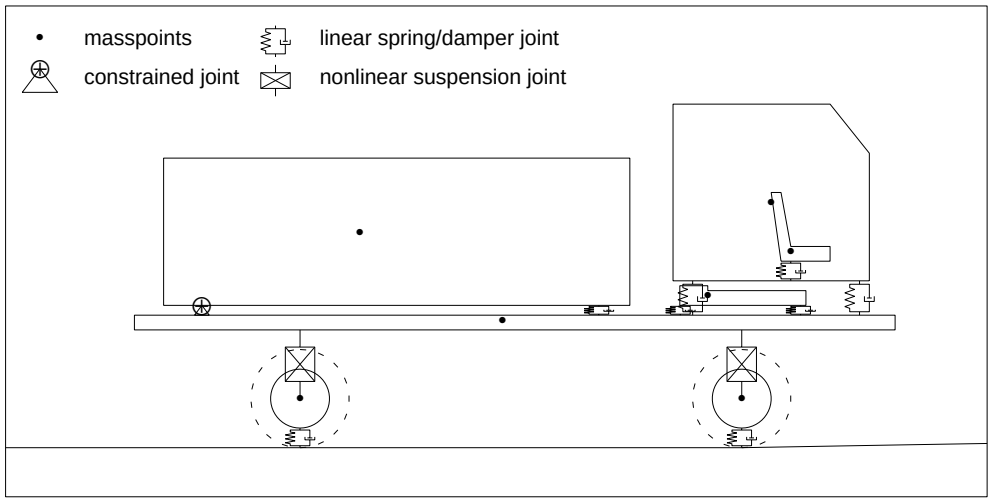


Figure 5.2: Planar truck model

This test problem has been used to study the response of the chassis and of the driver seat to excitations of the road. In [33], two road profiles have been used. First, a simple sine excitation and second, more realistic, a Fourier approximation of a driveway measurement. First, we solve both problems in the index–2 formulation with the standard MEXX code. The dimensions are $n_p = n_v = 11$, $n_g = n_\lambda = 1$. In Table 5.3 the results for some required tolerances are summarized. Furthermore, for the tolerance $RTOL = ATOL = 10^{-5}$, we solve additionally with the symmetric and sparse linear algebra option switched on. Again, as for the seven body mechanism, the integration is slightly affected by the different roundoff in the linear system solution. Although the dimension of the linear system is comparatively small, there is no efficiency loss due to the sparse mode solution.

Analysing the performance of MEXX in more detail reveals that for this problem the explicit discretization (2.1) is at the limit of its applicability. The stiff spring/damper elements used in this model make the equations of motion, at least, mildly stiff. Within the course of integration this can be observed especially in the last phase of the realistic excitation, where the excitation becomes smaller and smaller and, finally, vanishes totally. To

maintain stability, the integrator is forced to make small time steps also in this region, although there is no longer dynamics in the system. Depending on the length of such a "smooth" phase, the application of an implicit discretization may be more efficient.

This can be seen by solving the set of reduced equations of motion where the constraint equation is removed. This implicit system of ODEs, again from [33], can be solved with a standard stiff ODE solver. We used the extrapolated semi–implicit Euler discretization (code EULSIM [9]). A straightforward application (e.g. standard numerical differentiation for the Jacobian evaluation) shows the following results. For the sine excitation, where the dynamics of the problem forces small steps for the whole integration interval, the stiff integrator is faster only for $TOL = 10^{-3}$. For the realistic excitation, however, EULSIM is more efficient for the tolerances $TOL = 10^{-3}/10^{-5}$.

Excitation	Required Tolerance	Linear Algebra	*nstep*	*nfcn*	*nsol*	CPU(s)
measure	10^{-3}	full	692	9492	10185	142.4
	10^{-5}	full	552	16489	17042	186.8
		symm.	553	16491	17045	171.8
		sparse	551	16387	16939	168.2
	10^{-7}	full	511	32772	33284	331.0
	10^{-9}	full	595	53904	54500	545.9
	10^{-11}	full	766	81473	82240	817.3
sine	10^{-3}	full	301	5749	6051	62.3
	10^{-5}	full	274	13540	13815	84.5
		symm.	274	13540	13815	82.6
		sparse	277	13341	13619	78.9
	10^{-7}	full	304	25200	25505	144.2
	10^{-9}	full	349	38282	38632	218.2
	10^{-11}	full	492	54255	54748	310.8

Table 5.3: Results of MEXX solving the model with measured excitation and with sine excitation

5.4 Insulator Chain

To ensure sufficient safety in overland high voltage lines, the conducting cables have to be suspended on the poles by two chains of insulators. If one chain breaks, then the remaining one has to withstand the increased stress. The safe design of insulator chains is aided by dynamic simulation of the motion and the constraint forces occurring after fracture of one part of a double (or possibly triple) chain of insulators [17, 25].

Consider the double chain of cap insulators as sketched in Figure 5.3.

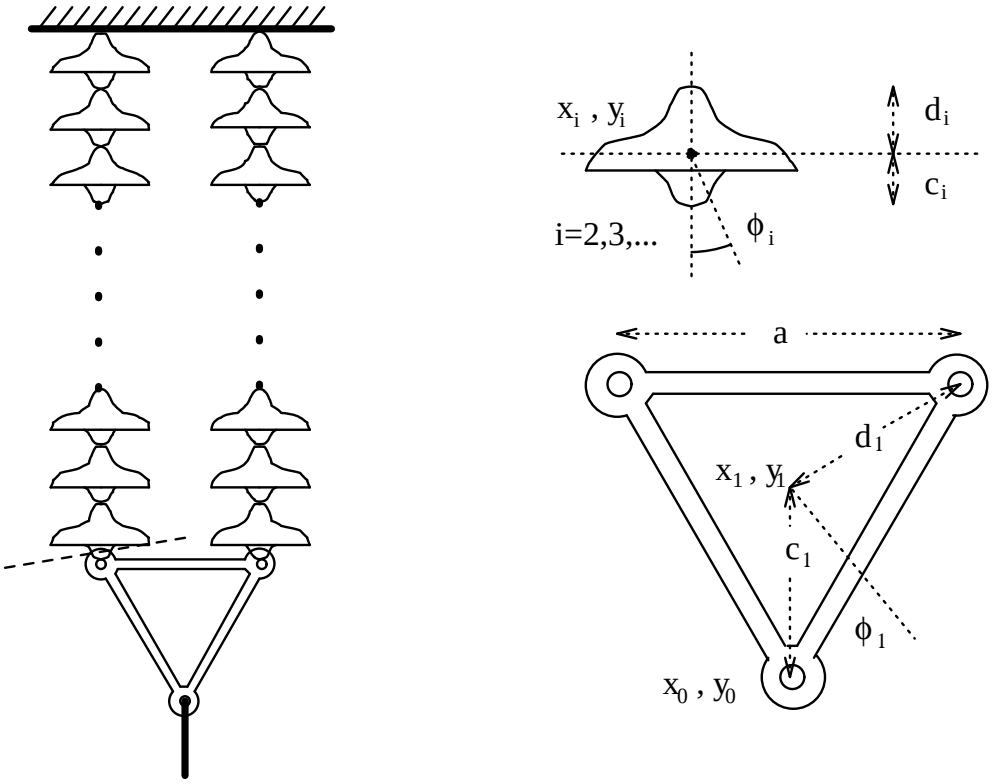


Figure 5.3: Chain of cap insulators

When the configuration breaks, e.g. at the joint between one of the chains and the triangular distance–holder, then the force exerted by the cable leads to a rapid movement of the remaining chain and a sudden increase of the constraint forces in the joints. Under the modelling assumptions of [17, 25], we get the following equations of motion which are most conveniently formulated as a set of differential–algebraic equations of the type (5.1).

To fix notation, let N denote the number of connected insulators. The coordinates of their centers and the position angles are given by $x_i, y_i, \varphi_i, i = 2, \ldots, N+1$. We denote the position and orientation of the triangle by x_1, y_1, φ_1. Finally, the force application point (at the triangle body) is denoted by x_0, y_0. Then, the dynamical equation (without reaction forces $G^T\lambda$) read:

For the force application point

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \ddot{x}_0 \\ \ddot{y}_0 \end{pmatrix} = F \begin{pmatrix} \sin\beta \\ -\cos\beta \end{pmatrix} \tag{5.6}$$

where

$$\begin{aligned} F &= F_0 + \dot{y}_0 \cdot EA/C_L \\ \beta &= -\arctan \tfrac{1}{C_Q}\dot{x}_0 \ . \end{aligned} \tag{5.7}$$

For the insulators and the triangle body

$$\begin{pmatrix} m_i & & \\ & m_i & \\ & & I_i \end{pmatrix} \begin{pmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{\varphi}_i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \qquad i = 1, \ldots, N+1 \ . \tag{5.8}$$

The constraints are:

Force application point at triangle corner

$$0 = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} - \left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} - d_1 \begin{pmatrix} -\sin\varphi_1 \\ \cos\varphi_1 \end{pmatrix} \right) \ . \tag{5.9}$$

Connection triangle corner with bottom insulator

$$\begin{aligned} 0 = \begin{pmatrix} g_3 \\ g_4 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \tfrac{d_1}{2} \begin{pmatrix} \cos\varphi_1 & -\sin\varphi_1 \\ \sin\varphi_1 & \cos\varphi_1 \end{pmatrix} \begin{pmatrix} \sqrt{3} \\ 1 \end{pmatrix} \\ - \left(\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - c_2 \begin{pmatrix} -\sin\varphi_2 \\ \cos\varphi_2 \end{pmatrix} \right) \ . \end{aligned} \tag{5.10}$$

Connections between the insulators

$$0 = \begin{pmatrix} g_{2i+1} \\ g_{2i+2} \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + d_i \begin{pmatrix} -\sin\varphi_i \\ \cos\varphi_i \end{pmatrix} - \left(\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} - c_{i+1} \begin{pmatrix} -\sin\varphi_{i+1} \\ \cos\varphi_{i+1} \end{pmatrix} \right) \qquad i = 2, \ldots, N \ . \tag{5.11}$$

Connection of top insulator with suspension point

$$0 = \begin{pmatrix} g_{2N+3} \\ g_{2N+4} \end{pmatrix} = \begin{pmatrix} x_{N+1} \\ y_{N+1} \end{pmatrix} + d_{N+1} \begin{pmatrix} -\sin\varphi_{N+1} \\ \cos\varphi_{N+1} \end{pmatrix} \ . \tag{5.12}$$

Thus, with the position vector

$$p := (x_0, y_0, x_1, y_1, \phi_1, ...x_{N+1}, y_{N+1}, \varphi_{N+1})^T \tag{5.13}$$

we have again a constrained mechanical system of the form (5.1) with dimensions

$$n_p = n_v = 3 \cdot (N+1) + 2 \ , \quad n_\lambda = n_g = 2 \cdot (N+1) + 2 \tag{5.14}$$

and functions

$$\begin{array}{rcl} M & = & diag(0, 0, m_1, m_1, I_1, \ldots, m_{N+1}, m_{N+1}, I_{N+1}) \\ f & = & (f_{0,x}, f_{0,y}, 0, 0, 0, \ldots, 0, 0, 0)^T \\ g & = & (g_1, g_2, \ldots, g_{2N+3}, g_{2N+4})^T \\ g^I & = & \partial g / \partial t = 0 \\ G & = & \partial g / \partial p \ . \end{array} \tag{5.15}$$

Fairly realistic values of the physical parameters are as follows: $N = 32$

$$\begin{array}{lll} m_1{=}34, & I_1{=}3.1, & d_1{=}0.37 \\ m_i{=}15, & I_i{=}0.35, & d_i{=}0.12, \quad c_i{=}0.08, \ i = 2, \ldots, N \\ m_j{=}9.8, & I_j{=}0.05, & d_j{=}0.16, \quad c_j{=}0.08, \ j = N+1 \\ C_L{=}\sqrt{E/\rho}, \ C_Q{=}\sqrt{F_0/(A\rho)} & & \\ E{=}8 \cdot 10^9, \quad A{=}3.4 \cdot 10^{-4}, \ \rho{=}3325, \ F_0{=}230000 \ . & & \end{array} \tag{5.16}$$

Consistent initial values for position and velocity are

$$\begin{array}{rcl} \varphi_1 & = & 0 \ , \ \ x_i, y_i \ \text{ such that (5.9–5.12) holds} \\ v & = & \dot{p} = 0 \ . \end{array} \tag{5.17}$$

Recall that it is not necessary to derive consistent initial values for the acceleration vector or for the Lagrange multipliers, respectively, as they are not needed for the integration. Sufficiently correct values (for dense output) may be computed within MEXX by switching on the corresponding option.

Figure 5.4 shows the simplified shape of the remaining insulator chain after breaking–off between one chain and the triangle.

The original number of insulators to be simulated is given by $N = 32$. Now, in order to check the performance of the linear algebra routines, we also present results for a system with, approximately, the "halfed" and "doubled" dimension, i.e. we also use $N = 16$ or $N = 64$ respectively. All interesting dimensions are summarized in Table 5.4. Herein, "$dim(A)$" indicates the dimension of the matrix to be decomposed,

$$A = \begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \ , \tag{5.18}$$

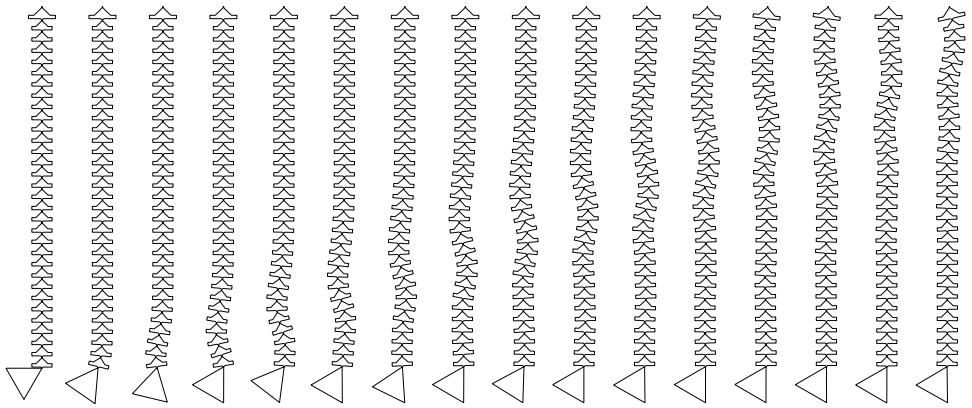


Figure 5.4: Shape of the remaining insulator chain at 0.0 up to 0.15 seconds after breaking–off in steps of 0.01 seconds

N	$n_p = n_v$	$n_\lambda = n_g$	$dim(A)$	$nne(A)$
16	53	36	89	327
32	101	68	169	631
64	197	132	329	1239

Table 5.4: Dimensions for the different insulator problems

whereas in the last column "$nne(A)$" the number of structural nonzero elements is given. For all comparison runs a final time $t_{end} = 0.1$ was chosen.

The interesting question is whether the sparse matrix mode for the linear system solution does now pay off. A summary of solving the three systems (halfed, standard and doubled) is given in Table 5.5.

Again, we show the required CPU time for the different linear algebra modes. Confirming the observations made so far, the symmetric full mode variant is less efficient than the general full mode variant. As expected, the general sparse mode LU–decomposition/substitution is drastically faster than the full mode variants. For all runs presented in the table the performance of the integration is not disturbed by the different roundoff errors of the linear algebra computation.

A quite interesting result shows the comparison of the increased CPU time with the increased dimension of the problem. As the absolute numbers of Table 5.5 cannot be compared directly due to the different behaviour of MEXX solving the different problems, one may look to the CPU time which is required on an average for one decomposition or solution respectively.

Required Tolerance	steps *nstep*	F-eval *nfcn*	Solves *nsol*	CPU(s) sparse	lin. alg. CPU(s) sparse	full	symm.
10^{-3}	13	583	597	20.1	16.9	89.5	108.9
10^{-5}	13	846	862	28.5	24.3	128.9	155.7
10^{-7}	17	1396	1414	47.0	39.6	211.9	265.1
10^{-3}	13	473	488	30.4	25.5	258.2	314.2
10^{-5}	12	759	774	47.8	40.7	410.2	496.7
10^{-7}	15	1204	1220	75.3	63.8	653.2	784.3
10^{-3}	13	444	460	61.5	52.8	935.4	1116.0
10^{-5}	15	890	906	121.5	104.1	1856.4	2212.4
10^{-7}	15	1164	1180	145.0	124.9	2418.9	2907.0

Table 5.5: Results for MEXX simulating 16/32/64 insulators

These numbers are given in Table 5.6. The values show, that doubling the dimension increases the average CPU time by a factor less or equal to 2 for the sparse mode routines. As expected, the increase for the full mode substitution is about a factor of 4. But, surprisingly, the increase factor is not cubed for the n^3 process of a full mode LU–decomposition. Rather, the factor is again approximately 4.

mode	subrout.	16	32	64
full	DGEFA	0.131	0.466	1.800
	DGESL	0.019	0.068	0.248
symm	DSIFA	0.163	0.582	2.242
	DSISL	0.018	0.060	0.209
sparse	MA28AD	0.068	0.132	0.287
	MA28BD	0.022	0.041	0.086
	MA28CD	0.0041	0.0074	0.016

Table 5.6: Average CPU time for decomposition and solution

This linear increase of computing time for the sparse linear algebra mode with respect to the dimension of the matrix may be compared to the amount of work needed for a simulation in minimal coordinates. Hereby the dimension of the matrices to be decomposed is smaller than in the descriptor formulation but the sparse structure is lost. The matrices are dense, thus the amount of work for the linear system solution is now proportional to N^3, where N denotes the number of bodies.

As a final experiment, the problem is now solved up to a final time $t_{end} = 0.2$ and with both, the dense output option and the root finding option, switched on. For the dense output requirements we choose only the position vector to be interpolated at 5 equidistant output points within each integration interval $[t, t + H]$. Concerning the root finding option, the following requirement was made. Find (and stop at) that time point where the top insulator shows its (first) maximal displacement. Thus we choose

$$\dot{v}_{101} = \dot{\varphi}_{N+1} = 0$$

as our switching function. Note that during the first phase of the simulation this condition holds, at least within the frame of the required tolerance – again $RTOL = ATOL = 10^{-5}$. But, as pointed out in the preceding section, the root finding algorithm is started only, if at both endpoints of the current interval the value of the switching function is "not small".

Compared to a run with both options switched off and $t_{end} := t_{root} = 0.1283$ (overall CPU time: 71.0s) the required CPU time (83.7s) increases by 18 %. The main part of this additional time is needed for the dense output generation, i.e. for the generation of the extrapolated divided differences, cf. Section 2.4. The root finding process is nearly for free, as only one root has to be determined. The evaluation of the interpolating polynomial turns out to be quite cheap (0.45s), although 101 components at 80 interpolation points (and all 338 components at some points for root finding) have to be evaluated.

Acknowledgements. The authors want to express their sincere thanks to P. Kaps for providing the insulator chain example, to B. Simeon for sending the FORTRAN programs for the truck model, and to G. Leister for sending those of the seven–body mechanism.

References

[1] T. Alishenas: *Zur numerischen Behandlung, Stabilisierung durch Projektion und Modellierung mechanischer Systeme mit Nebenbedingungen und Invarianten.* Dissertation, TRITA–NA–9202, Royal Inst. Techn., Stockholm (1992)

[2] D.S. Bae, E.J. Haug: *A recursive formulation for constrained mechanical system dynamics. I and II.* Mech. Struct. & Mach., **15**, 359–382 and 481–506 (1987)

[3] A. Björck: *Least squares methods,* in P.G. Ciarlet, J.L. Lions (eds.): *Handbook of Numerical Analysis, Vol. I.* North–Holland, Amsterdam (1990)

[4] H. Brandl, R. Johanni, M. Otter: *A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix.* Proc. IFAC/IFIP/IMACS Intl. Symposium on Theory of Robots, Vienna (1986)

[5] V. Brasey: *A half–explicit Runge–Kutta method of order 5 for solving constrained mechanical systems.* Comput. **48**, 191–201 (1992)

[6] H. Brauchli, R. Weber: *Dynamical equations in natural coordinates.* Comp. Meth. Appl. Mech. Eng. **91**, 1403–1414 (1991)

[7] K.E. Brenan, S.L. Campbell, L.R. Petzold: *Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations.* North–Holland, New York (1989)

[8] P. Deuflhard: *Order and stepsize control in extrapolation methods.* Numer. Math., **41**, 399–422 (1983)

[9] P. Deuflhard: *Recent Progress in Extrapolation Methods for ODE's.* SIAM Review 27 (1985), p. 505–535

[10] J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart: *LINPACK.* SIAM, Philadelphia (1979)

[11] I.S. Duff, U. Nowak: *On Sparse Solvers in a Stiff Integrator of Extrapolation Type.* IMA Journal of Numerical Analysis **7**, p. 391–405 (1987)

[12] I.S. Duff: *MA28 – A set of FORTRAN Subroutines for Sparse Unsymmetric Linear Equations.* AERE Report R. 8730, HMSO, London (1977)

[13] I.S. Duff, N.I.M. Gould, J.K. Reid, J.A. Scott, K. Turner: *The factorization of sparse symmetric indefinite matrices.* IMA J. Numer. Anal., **11**, 181–204 (1991)

[14] E. Eich: *Projizierende Mehrschrittverfahren zur numerischen Lösung von Bewegungsgleichungen technischer Mehrkörpersysteme mit Zwangsbedingungen und Unstetigkeiten.* Fortschr.–Ber. VDI Reihe 18, **109**, (1992)

[15] Ch. Engstler: Diploma thesis, Universität Innsbruck, in preparation.

[16] R. Fletcher: *Practical Methods of Optimization.* Wiley, Chichester, New York (1987)

[17] P. Hagedorn, H. Idelberger, L. Möcks: *Dynamische Vorgänge bei Lastumlagerung in Abspannketten von Freileitungen.* etz Archiv 2, p. 109–119 (1980)

[18] E. Hairer, Ch. Lubich: *Extrapolation at stiff differential equations.* Numer. Math., **52**, 377–400 (1988)

[19] E. Hairer, Ch. Lubich, M. Roche: *The Numerical Solution of Differential–Algebraic Systems by Runge–Kutta Methods.* SLNM **1409**, Springer–Verlag, Berlin (1989)

[20] E. Hairer, S.P. Nørsett, G. Wanner: *Solving Ordinary Differential Equations I. Nonstiff Problems.* Springer–Verlag, Berlin (1987)

[21] E. Hairer, G. Wanner: *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems.* Springer–Verlag, Berlin (1991)

[22] E. Hairer, A. Ostermann: *Dense output for extrapolation methods.* Numer. Math., **58**, 419–439 (1990)

[23] E.J. Haug: *Computer Aided Kinematics and Dynamics of Mechanical Systems. Volume I: Basic Methods.* Allyn & Bacon, Boston (1989)

[24] Ch. Lubich: *Extrapolation integrators for constrained multibody systems.* IMPACT Comp. Sci. Eng., **3**, 213–234 (1991)

[25] J. Müller: *Dynamische Vorgänge bei Brüchen in Isolator–Doppelketten von Freileitungen.* Österr. Ing. Arch. Z. **136**, 569–579 (1991)

[26] C.C. Paige: *An error analysis of a method for solving matrix equations.* Math. Comp., **27**, 355–359 (1973)

[27] L.R. Petzold, P. Lötstedt: *Numerical solution of nonlinear differential equations with algebraic constraints. II. Practical implications.* SIAM J. Sci. Stat. Comp., **7**, 720–733 (1986)

[28] F. Potra: *Multistep methods for solving constrained equations of motion.* Report, Dept. of Math., Univ. of Iowa (1991)

[29] R.E. Roberson, R. Schwertassek: *Dynamics of Multibody Systems.* Springer–Verlag, Berlin (1988)

[30] W. Schiehlen: *Technische Dynamik.* Teubner Studienbücher Mechanik, Teubner Verlag, Stuttgart (1986)

[31] W. Schiehlen (ed.): *Multibody Systems Handbook.* Springer–Verlag, Berlin (1990)

[32] R. Schwertassek, A. Eichberger: *Recursive generation of multibody system equations in terms of graph theoretic concepts.* Mech. Struct. & Mach., **17**, 197–218 (1989)

[33] B. Simeon, F. Grupp, C. Führer, P. Rentrop: *A Nonlinear Truck Model and its Treatment as a Multibody System.* Report TUM–MATH-05-92-MO4-250/1.-FMI, Technische Universität München, (1992)

[34] B. Simeon, C. Führer, P. Rentrop: *Differential–algebraic equations in vehicle dynamics.* Surv. Math. Ind., **1**, 1–37 (1991)

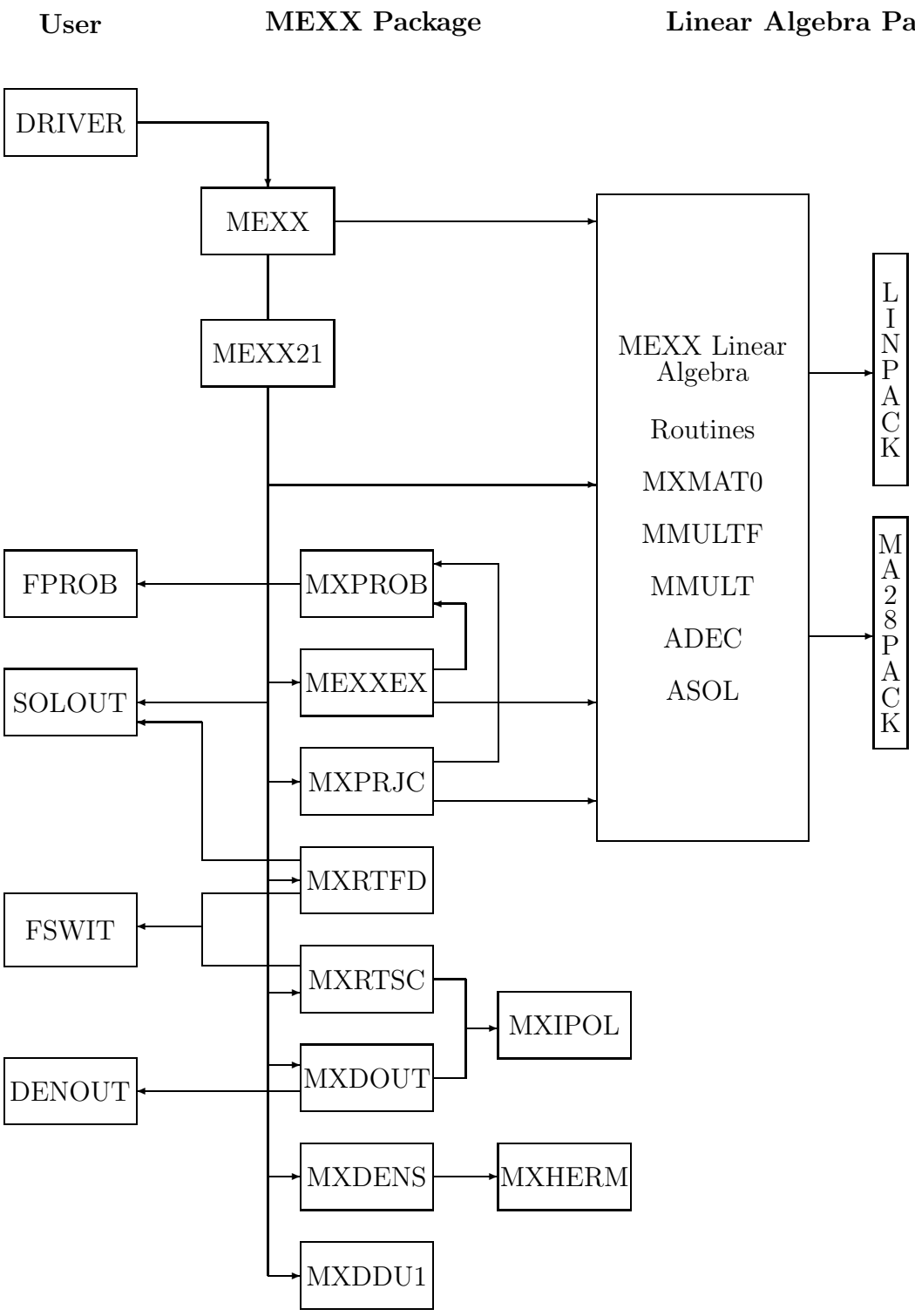
[35] R.A. Wehage: *Application of matrix partitioning and recursive projection to O(n) solution of constrained equations of motion.* Report (1989)

[36] R.A. Wehage, E.J. Haug: *Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic systems.* J. Mech. Design, **134**, 247–255 (1982)

[37] J. Wittenburg: *Dynamics of Systems of Rigid Bodies.* Teubner, Stuttgart (1977)

Appendix A

Program Structure



Routine	Purpose
MEXX	the user interface routine with input check, workspace distribution, and initialization of the linear algebra
MEXX21	controls the numerical integration, the projection, and the dense output generation
MXPROB	an interface routine between the MEXX package and the user written problem routine
MEXXEX	performs one integration step with extrapolation and error check
MXPRJC	performs the projection step
MXRTFD	root finder
MXRTSC	checks for sign change in the user written switching function
MXDOUT	generates the dense output solution values according to the selected options
MXDENS	computes extrapolated divided differences
MXDDU1	dumps data for dense output postprocessing
MXIPOL	computes interpolating values
MXHERM	computes the coefficients for hermite interpolation
MXMAT0	performs linear algebra for MEXX

Table A.1: Purpose of MEXX subroutines

Entry	Purpose
MXMAT0	initialize workspace for linear algebra
MMULTF	computes a matrix vector product
MMULT	computes another matrix vector product
ADEC	interface to matrix decomposition, calls appropriate LINPACK or MA28 subroutines
ASOL	interface to linear system solution based on the decomposition done by ADEC, calls appropriate LINPACK or MA28 subroutines

Table A.2: Entries of the linear algebra subroutine MXMAT0

Routine	Purpose
DRIVER	main program
FPROB	called to evaluate the different functions describing the problem
SOLOUT	called at integration points and at switching points providing the solution values
FSWIT	the user may define one or more functions and MEXX will detect its/their zeros and provide interpolated solution values at those zeros
DENOUT	called with interpolated solution values at user selected points of time

Table A.3: Routines to be supplied by the user of MEXX

Authors' addresses

Christian Engstler
Institut für Mathematik und Geometrie
Universität Innsbruck
Technikerstr. 13
A–6020 Innsbruck
e–mail: engstler@mat1.uibk.ac.at

Christian Lubich
Institut für Angewandte Mathematik und Statistik
Universität Würzburg
Am Hubland
D–8700 Würzburg
e–mail: lubich@mathematik.uni-wuerzburg.de

Ulrich Nowak und Uwe Pöhle
Konrad–Zuse–Zentrum für Informationstechnik Berlin
Heilbronner Strasse 10
D–1000 Berlin 31
e–mail: nowak@sc.zib-berlin.de und poehle@sc.zib-berlin.de