Acta Cybernetica **24** (2020) 315–341.

Multibody Dynamics in Natural Coordinates through Automatic Differentiation and High-Index DAE Solving[*]

John D. Pryce[a] and Nedialko S. Nedialkov[b]

Abstract

The Natural Coordinates (NCs) method for Lagrangian modelling and simulation of multibody systems is valued for giving simple, sparse models. We describe our version of it and compare with the classical approach of Jalón and Bayo (JBNCs). Our NCs use the high-index differential-algebraic equation solver DAETS. Algorithmic differentiation, not symbolic algebra, forms the equations of motion from the Lagrangian. We obtain significantly smaller equation systems than JBNCs, at the cost of a non-constant mass matrix for fully 3D models—a minor downside in the DAETS context. Examples in 2D and 3D are presented, with numerical results.

Keywords: Lagrangian mechanics, differential-algebraic equations, natural coordinates, simulation, algorithmic differentiation, YAML

1 Introduction

1.1 Context and aims

We are concerned with simulating multibody systems (MBS, aka mechanisms), built mainly from rigid bodies, with joints and other ways to interact. Recall some advantages of a Lagrangian approach [6, 20] to forming their equations of motion.

Economy. Lagrangians, in contrast to direct use of Newton's three laws, can omit mention of forces that do no work, e.g. reaction force of smooth sliding contact.

Flexibility. One is free to choose *generalised coordinates* $\mathbf{q} = (q_1, \dots, q_{n_q})$ to specify system position. Among all possible motions of the system, the actual one is a stationary point of the action integral $\int \mathcal{L} dt$ of the Lagrangian function $\mathcal{L} = T - V$, where $T(t, \mathbf{q}, \dot{\mathbf{q}})$ and $V(t, \mathbf{q})$ are the system's kinetic and potential

[*]N. Nedialkov was funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

[a]Cardiff University - corresponding author, E-mail: `PryceJD1@cardiff.ac.uk`

[b]McMaster University, E-mail: `nedialk@mcmaster.ca`

DOI: 10.14232/actacyb.24.3.2020.4

energies. This is an inherent system property, so the equations of motion that come from the Euler–Lagrange variational conditions:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial \mathcal{L}}{\partial \dot{q}_j} - \frac{\partial \mathcal{L}}{\partial q_j} + \sum_{i=1}^{n_c} \lambda_i \frac{\partial C_i}{\partial q_j} = Q_j(t), \quad j = 1, \dots, n_q, \tag{1}$$

$$C_i = 0, \quad i = 1, \dots, n_c, \tag{2}$$

describe the same set of possible motions, independent of the chosen $\mathbf{q}$. The n_c equations (2) are constraints on the motion due to rigidity etc., with associated Lagrange multipliers λ_i. The $Q_j(t)$ are *generalised external force* components if any, whose definition also involves $\partial/\partial q_j$. We assume *holonomic* constraints, i.e. independent of velocities, so of the form $C_i(t, \mathbf{q}) = 0$; the positional and velocity degrees of freedom are then equal:

$$\textsc{dof} = n_q - n_c. \tag{3}$$

So the system state is locally fixed at any t by DOF values q_j and DOF values $\dot{q}_j$.

If constraints are absent ($n_c = 0$), then (1) is reducible to an ordinary differential equation system (ODE). If present ($n_c > 0$), it is a differential-algebraic equation system (DAE)—usually true when $\mathbf{q}$ consists of cartesian coordinates, because rigid-body constraints must be included. It has index 3, in the classical differential index sense [1], when solved as an initial-value problem. High-index DAEs have been seen as hard to solve numerically, so much effort has gone into making $n_c = 0$, i.e. finding representations that lead to constraint-free coordinates, typically angles.

We argue firstly, that an efficient *high-index DAE solver* changes the balance of advantage, thereby encouraging the use of *Natural Coordinates* with their benefits of simple, sparse, human-readable equations. We use the `C++` code DAETS (DAE by Taylor Series) [11, 12] that can integrate numerically arbitrary index DAEs.

Secondly, we avoid *symbolic manipulation* to convert the Lagrangian $\mathcal{L}$ into the equations of motion (1) given to a numerical solver. DAETS's built in *algorithmic differentiation* (AD) system does this at run time—the *Lagrangian facility* [19].

Thirdly, only shown briefly by examples here, the *mechanism facility* reads at run time a "MechSpec" text file describing a mechanism, and constructs from it a Natural Coordinates model of the Lagrangian and constraints, which is passed to the Lagrangian facility and thence to DAETS for numerical solution.

This article is about purely continuous systems and does not touch on the large area of hybrid systems, with a mixture of continuous and discrete-event behaviour.

In the following, Section 2 is about Natural Coordinates: §2.1 describes differences between other versions of NCs and ours, and §2.2 presents ours in detail. Section 3 outlines our numerical infrastructure. Section 4 gives to 2D and 3D examples, with numerical results. Section 5 summarises the effect of our method's theory and software architecture, and discusses the resulting system's friendliness, in particular in a teaching context.

2 Natural Coordinates

2.1 Our method compared with others

As with other Natural Coordinates (NCs) methods, our $\mathbf{q}$ holds mainly cartesian coordinates in the world-frame (WF) of *points* or *vectors* (PVs) fixed on mechanism parts. We compare with the 1994 de Jalón and Bayo presentation (JBNCs) in [4] and later ones by von Schwerin [22] and Kraus *et al.* [8]; see also the surveys by Nikravesh (2004) [15], and de Jalón (2007) [3].

Four-PVs versus three-PVs NCs. Maybe influenced by classical finite elements, these authors put emphasisis on computing a mechanism's *mass matrix* $\mathbf{M}$ and on methods that lead to a constant $\mathbf{M}$, independent of system position.

This leads them, see [4, pp. 44–51], [22, p. 54], [8, end of §1], to define a general 3D rigid body's position by four PVs—i.e. 12 scalar coordinates. E.g. from [3, §1] *"When all the bodies contain at least two points and two unit vectors the inertia matrix is constant and there aren't velocity dependent inertia forces".*

A free rigid body has 6 DOF so there must be $12-6=6$ scalar constraints to impose rigidity, per body. For us, constant $\mathbf{M}$ is nearly irrelevant because of the underlying algorithm architecture, see below. Hence, we choose three PVs, 9 scalar coordinates, to define a body's position. In JBNC terminology, these are *basic* PVs, or BPVs. To produce 6 DOF, we use $9 - 6 = 3$ rigidity constraints per body.

Frame definition and rigidity. Three BPVs determine the origin and x, y directions of an orthonormal frame. We define the z direction by a cross product—a nonlinear operation, which is what makes our $\mathbf{M}$ a function of $\mathbf{q}$ in general. In 2D $\mathbf{M}$ is always constant, and for simple mechanisms in 3D can often be made so by a good choice of BPVs. The cross product makes our frames positively oriented, which need not hold in the other approaches.

JBNC devote several pages [4, pp. 47–51] to setting up the frame and the rigidity constraints for various cases of points/vectors defining body position; and more (pp 130–143) to cases of computing the mass matrix. Mostly they aim at a constant $\mathbf{M}$, but include a case [4, p. 139, see Fig 4.7] of defining the frame by a cross product exactly as we do, leading to a varying $\mathbf{M}$. By contrast, we have one unified method, that treats point and vector frame elements in the same way. (An exception, that applies to all approaches, is the case of a dimensionally deficient part such as a particle or a thin rod, for which a 3D frame makes no sense.)

Kraus *et al.* [8, §4] trade increased model size for the benefit of increased simplicity and sparsity. They define frames in a unified way close to how our method would work if we used a "four PVs" rather than "three PVs" representation; if one makes a QR factorisation of their matrix X (the R is constant so can be precomputed; the Q is varying), the methods become very similar.

Von Schwerin does not give great detail of his NCs but [22, p. 48] says "one of the advantages of the natural coordinate approach is that the resulting mass matrix M is constant", suggesting he only considers a "four PVs" system.

Basic and dependent PVs. For us, points/vectors fixed to a body $\mathcal{R}$ are either basic or *dependent*. Basic ones, BPVs, are either *fixed* in the WF, or *moving*. Vector $\mathbf{q}$ comprises all moving BPVs[1], plus some needed scalars such as turn angles. We specify a dependent PV by giving its coordinates in the frame defined by the BPVs. It might be part of defining a joint, or a point where a force is applied to $\mathcal{R}$, etc.

Since each BPV is either fixed or a component of $\mathbf{q}$, each dependent PV is an explicit function of $\mathbf{q}$. Jalón–Bayo [4, p. 51] describe this dependent point method, but merely as a useful trick for bodies with many PVs, not as a primary technique.

Dynamics data. We specify $\mathcal{R}$'s *dynamics* by storing the local coordinates of the centroid, the mass, and the inertia matrix about the centroid. This seems more natural than the method, described in [15, §4.4], of "lumping" the mass into virtual particles at the BPVs in such a way as to preserve the inertial properties.

Assembly process. We assemble $\mathcal{L}$ one part at a time. Think of the method as a function whose input is the values $\mathbf{q}, \dot{\mathbf{q}}$ at current time and whose output is $\mathcal{L}$, plus rigidity and joint constraints as a vector of residual values to be driven to zero. Fixed PVs are global constants, dependent PVs are local variables (temporaries).

For each part, in a loop, use $\mathbf{q}, \dot{\mathbf{q}}$ to find its current WF position and linear and angular velocity. Using its dynamics data, find the part's kinetic and where relevant its potential energy and accumulate these into $\mathcal{L}$. Include (the residuals of) the part's rigidity equations in the constraints array.

Similarly loop over joints to add their constraints, over any springs to add their PE contributions, and so on.

The objects to which this is all done are not numbers but Taylor series. The output is passed to the Lagrangian facility, which uses automatic differentiation to do the needed $\mathrm{d}/\mathrm{d}t$, $\partial/\partial\mathbf{q}$ and $\partial/\partial\dot{\mathbf{q}}$ that converts them into the Euler–Lagrange DAE in a form DAETS can handle.

Relevant matrices. The mass matrix $\mathbf{M}$ is not mentioned in the above steps. However DAETS produces for any DAE, from analysing its structure, a numerical *system Jacobian* matrix $\mathbf{J}$ that is central to its Taylor expansion process. In our context, the DAE was created by the Lagrangian facility, so $\mathbf{J}$ is formed by the interplay between the latter and DAETS. $\mathbf{J}$ turns out to be exactly the symmetric matrix $\left[\begin{smallmatrix}\mathbf{M} & \mathbf{G}^T \\ \mathbf{G} & \mathbf{0}\end{smallmatrix}\right]$ that is ubiquitous in numerical treatment of MBS. Here $\mathbf{M}$ is the mass matrix and $\mathbf{G}$ is the constraint Jacobian. In general, for our method both $\mathbf{M}$ and $\mathbf{G}$ depend on $\mathbf{q}$. But each instance of $\mathbf{J}$ is re-used 12 to 20 times (to solve linear systems) as part of the high-order Taylor series expansion, so the cost of evaluating and factorising it is well amortised.

Descriptor form. This is a point of theory that seems to affect how MBS are solved in practice. The Euler–Lagrange DAE (1, 2) is usually manipulated into the *descriptor form*, e.g. as given in [22, p. 25, eq. (1.2.8a,b)]

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

[1]That always works, but one can often get a smaller $\mathbf{q}$ via dependency analysis.

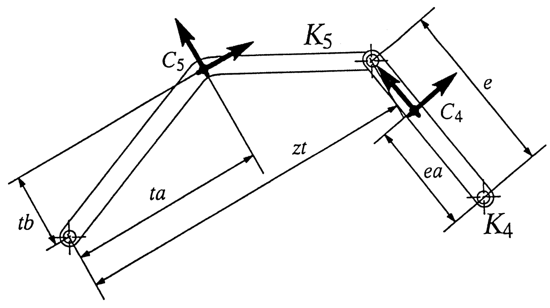


Table 9.1. Geometrical parameters

$d = 0.028$	$da = 0.0115$	$e = 0.02$
$ea = 0.01421$	$zf = 0.02$	$fa = 0.01421$
$rr = 0.007$	$ra = 0.00092$	$ss = 0.035$
$sa = 0.01874$	$sb = 0.01043$	$sc = 0.018$
$sd = 0.02$	$zt = 0.04$	$ta = 0.02308$
$tb = 0.00916$	$u = 0.04$	$ua = 0.01228$
$ub = 0.00449$	$c_0 = 4530$	$\ell_0 = 0.07785$

Fig. 9.2. Geometrical design (Schiehlen 1990, with permission)

Figure 1: From [7, §VI.9]. Straight rod K_4, bent rod K_5 are two of ASM's 7 parts, drawn with local frames based at C_4, C_5. Table on right gives lengths ta, tb etc.

and similarly p. 124 Example 4.1 and p. 158 eq. (5.4) in [4]. The cost of bringing in $\ddot{\mathbf{q}}$ explicitly is that the right hand side involves forming $\dot{\mathbf{M}}$, which brings in quadratic terms $\dot{q}_i^2$ and $\dot{q}_i\dot{q}_j$ that are identified as centrifugal and Coriolis force terms respectively. Avoiding such terms is one reason why constant $\mathbf{M}$ is valued.

However one can formulate the system as a coupled first-order DAE in $(\mathbf{p}, \mathbf{q})$, where $\mathbf{p}$ is the vector of generalised momenta $\partial\mathcal{L}/\partial\dot{\mathbf{q}}$—as used in a Hamiltonian formulation, and see §4.1.2 below. Indeed [4, p. 130, eqs (4.28-29)] presents exactly this form and points out that $\dot{\mathbf{M}}$ is absent. We wonder why it is not used more as a basis for numerical methods.

The above point is not relevant to how DAETS works; but we note that the generalised accelerations $\ddot{\mathbf{q}}$, if needed, drop out of its Taylor series solution.

Other points.

1. Our method aids automatic conversion from a high-level description of a mechanism to the Lagrangian (as the MechSpec does) e.g. for representing existing mechanisms. For instance Figure 1 is part of the geometry description of the Andrews Squeezing Mechanism [7, 16]. This is mainly a list of coordinates of dependent points in local frames and transcribes simply to a MechSpec text file.
2. The issue of *sharing several points* between bodies needs some thought. Figure 2 is from Nikravesh [15]. Here item b) shows bodies sharing two points, call them A, B at a distance L apart, creating a revolute joint. In the NC systems, the author describes, rigidity is enforced purely by length constraints—here the equation $(\mathsf{B}-\mathsf{A})\cdot(\mathsf{B}-\mathsf{A}) = L^2$, which is only given once. In our method, the most general case is that each of A and B is a dependent point in the frame of either body, so distance L is not explicitly stated but can be found by a Pythagorean calculation $L = \sqrt{\sum_{k=1}^{3}(A_k - B_k)^2}$. The computed values of L in the two bodies might disagree, owing to data error, or roundoff. We solve this by letting a point and a vector be shared between bodies, but not two points.

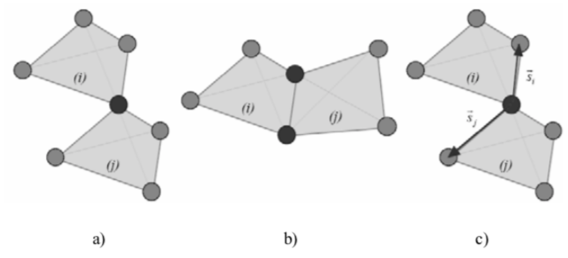


Figure 2: From [15, p. 192]

2.2 Our NCs in more detail

The world (fixed space) in which motion takes place is Euclidean space E^d of dimension d (2 or 3 in practice), with some point O chosen as origin. Identifying a point with its position-vector (henceforth just *point*) relative to O, it is a real inner product space. With a frame (orthonormal basis) chosen, it becomes the world-frame (WF). The style $\mathbf{q}$ is used for a generalised-coordinates vector; X for a *point* in a world or local frame; $\mathbf{x}$ for a *vector*, a difference of points. A rotation means a member of the group of orthogonal transformations on E^d with determinant $+1$.

Subscript $_t$, and the word "moving", mean "function of t", e.g. O_t is a moving point, synonymous with $\mathsf{O}(t)$. A rigid body $\mathcal{R}$ has its own local copy of E^d. Rigid motion of $\mathcal{R}$ is defined, with respect to a local frame and the WF, by

$$\mathcal{R}_t = \mathsf{O}_t + Q_t\mathcal{R}$$

for some moving point O_t and rotation matrix Q_t, which means by definition that each point X or vector $\mathbf{u}$ fixed in $\mathcal{R}$ moves in the WF according to

$$\mathsf{X}_t = \mathsf{O}_t + Q_t\mathsf{X}, \qquad \mathbf{u}_t = Q_t\mathbf{u}. \tag{4}$$

We now assume the 3D case till said otherwise.

Tracking position. As said in §2.1, we track the motion of $\mathcal{R}$ by an ordered list of three non-collinear BPVs fixed in $\mathcal{R}$. Denote them $\mathsf{O}, \mathsf{A}, \mathsf{B}$, but A and/or B might be a vector. They define the local frame as follows.
– O must be a point and is the local origin.
– A is a point or a vector and is on (along, if a vector) the local x axis.
– B is a point or a vector and is in/toward the local xy plane.
In the resulting local frame, the BPVs are columns of a matrix $\mathbf{P}$ of the form

$$\mathbf{P} = [\mathsf{O}, \mathsf{A}, \mathsf{B}] = \begin{bmatrix} 0 & r_{11} & r_{12} \\ 0 & 0 & r_{22} \\ 0 & 0 & 0 \end{bmatrix} = \left[\begin{array}{c|cc} 0 & \multicolumn{2}{c}{\multirow{2}{*}{R}} \\ 0 & & \\ \hline 0 & 0 & 0 \end{array}\right]. \tag{5}$$

A user specifies the frame by explicitly giving the coordinates of A, B—equivalently, the numbers r_{ij}. The 2×2 upper triangular matrix R must be nonsingular, $r_{ii} \neq 0$.

Usually $r_{ii} > 0$, but this need not be so, e.g. by setting $r_{11} < 0$, the user puts A on the negative side of the x axis.

Example 1. Let OA, OB have lengths 3 and $2\sqrt{2}$ and make an acute angle 45^o. We may give coordinates $\mathsf{A} = (3,0,0)^T$ along x direction, or $\mathsf{A} = (-3,0,0)^T$ in the opposite direction. Any of $(\pm 2, \pm 2, 0)^T$ can be coordinates of B. □

We form the unit vectors $\widehat{\mathbf{x}} = (1,0,0)^T$ and $\widehat{\mathbf{y}} = (0,1,0)^T$ in $\mathcal{R}$ as follows. In $\mathbf{P} = [\mathsf{O}, \mathsf{A}, \mathsf{B}]$, if A [resp. B] is a point, we must replace it by the vector $\mathsf{A} - \mathsf{O}$ [resp. $\mathsf{B} - \mathsf{O}$]. We write the subtraction(s) as $\mathbf{P}S$, where

$$S = \begin{bmatrix} -\alpha & -\beta \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \begin{array}{l} \alpha = \begin{cases} 1 & \text{if A is a point,} \\ 0 & \text{if A is a vector,} \end{cases} \\ \text{similarly for } \beta \text{ and B.} \end{array} \tag{6}$$

Denoting $U = SR^{-1}$, we have

$$[\widehat{\mathbf{x}}, \widehat{\mathbf{y}}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = \mathbf{P}U. \tag{7}$$

During motion, the time-varying matrix in the WF corresponding to (5) is

$$\mathbf{P}_t = [\mathsf{O}_t, \mathsf{A}_t, \mathsf{B}_t].$$

The properties of rigid motion (4) imply that the relation (7) in the local frame must hold in the WF for all t, that is

$$[\widehat{\mathbf{x}}_t, \widehat{\mathbf{y}}_t] = \mathbf{P}_t U = [\mathsf{O}_t, \mathsf{A}_t, \mathsf{B}_t]\, U; \tag{8}$$

$\widehat{\mathbf{x}}_t$, and $\widehat{\mathbf{y}}_t$ are the first two columns of Q_t. Its third column is, by the assumed positive orientation,

$$\widehat{\mathbf{z}}_t = \widehat{\mathbf{x}}_t \times \widehat{\mathbf{y}}_t. \tag{9}$$

We write (8, 9) compactly as follows. For a 3×2 matrix M, let $M \otimes$ be the 3×3 result of appending the cross product of its columns to it:

$$[\mathbf{u}, \mathbf{v}] \otimes = [\mathbf{u}, \mathbf{v}, \mathbf{u} \times \mathbf{v}]. \tag{10}$$

Combining (8, 9, 10), finding the moving Q_t from BPVs $\mathsf{O}_t, \mathsf{A}_t, \mathsf{B}_t$ is given, for all point/vector combinations of BPVs, by one formula

$$[\widehat{\mathbf{x}}_t, \widehat{\mathbf{y}}_t, \widehat{\mathbf{z}}_t] = Q_t = \big([\mathsf{O}_t, \mathsf{A}_t, \mathsf{B}_t]\; U\big) \otimes. \tag{11}$$

Dependent PVs in $\mathcal{R}$ are defined by giving their coordinates in the local frame. Once O_t and Q_t are found, we simply compute by (4). For instance, if $\overline{\mathsf{M}}$ is the (fixed) position of the centroid in the local frame, then at any time $\mathsf{M}_t = \mathsf{O}_t + Q_t \overline{\mathsf{M}}$.

Henceforth we drop the $_t$.

Example 2. Let $\mathcal{R}$ be a $30^o 60^o 90^o$ triangle ABC with long side AB of length 2. Define the frame by points $\mathsf{A} = \left[\begin{smallmatrix}0\\0\\0\end{smallmatrix}\right]$ and $\mathsf{B} = \left[\begin{smallmatrix}2\\0\\0\end{smallmatrix}\right]$ and a vector $\mathbf{c} = \left[\begin{smallmatrix}\sqrt{3}\\1\\0\end{smallmatrix}\right]$ in the direction $\overline{\mathsf{AC}}$. Then

$$S = \begin{bmatrix}-1 & 0\\ 1 & 0\\ 0 & 1\end{bmatrix}, \quad R = \begin{bmatrix}2 & \sqrt{3}\\ 0 & 1\end{bmatrix}, \quad \text{and} \quad U = SR^{-1} = \begin{bmatrix}-1/2 & \sqrt{3}/2\\ 1/2 & -\sqrt{3}/2\\ 0 & 1\end{bmatrix}.$$

At time t, we are given computed A, B and $\mathbf{c}$, and (8, 9) say

$$[\widehat{\mathbf{x}}, \widehat{\mathbf{y}}] = [\mathsf{A}, \mathsf{B}, \mathbf{c}]\, U = \left[-\tfrac{1}{2}\mathsf{A} + \tfrac{1}{2}\mathsf{B},\, \tfrac{\sqrt{3}}{2}\mathsf{A} - \tfrac{\sqrt{3}}{2}\mathsf{B} + \mathbf{c}\right], \quad \widehat{\mathbf{z}} = \widehat{\mathbf{x}} \times \widehat{\mathbf{y}}.$$

□

Tracking velocities and kinetic energy. Just as BPVs are in $\mathbf{q}$ or computable therefrom, the velocities $\dot{\mathsf{O}}, \dot{\mathsf{A}}, \dot{\mathsf{B}}$ are in $\dot{\mathbf{q}}$ or computable therefrom. To express $\mathcal{R}$'s KE in terms of them, differentiate (11). Eq. (8) is linear in the BPVs, so

$$[\dot{\widehat{\mathbf{x}}}, \dot{\widehat{\mathbf{y}}}] = [\dot{\mathsf{O}}, \dot{\mathsf{A}}, \dot{\mathsf{B}}]\, U,$$

and from (9), on differentiating the bilinear operation $\widehat{\mathbf{x}} \times \widehat{\mathbf{y}}$,

$$\dot{Q} = [\dot{\widehat{\mathbf{x}}}, \dot{\widehat{\mathbf{y}}}, \dot{\widehat{\mathbf{z}}}] \quad \text{where} \quad \dot{\widehat{\mathbf{z}}} = \dot{\widehat{\mathbf{x}}} \times \widehat{\mathbf{y}} + \widehat{\mathbf{x}} \times \dot{\widehat{\mathbf{y}}}. \tag{12}$$

Let $\overline{\mathsf{M}}$ be the constant position of $\mathcal{R}$'s centroid in its local frame. Then by (4, 12), the WF velocity $\dot{\mathsf{M}}$ of this centroid is

$$\dot{\mathsf{M}} = \dot{\mathsf{O}} + \dot{Q}\overline{\mathsf{M}}, \tag{13}$$

Recall that differentiating $Q^T Q = I$ shows $Q^T \dot{Q}$ is a skew-symmetric matrix equal to $\left[\begin{smallmatrix}0 & -\omega_3 & \omega_2\\ \omega_3 & 0 & -\omega_1\\ -\omega_2 & \omega_1 & 0\end{smallmatrix}\right]$, where ω_i are the components of the angular velocity vector $\boldsymbol{\omega}$ as seen from $\mathcal{R}$'s frame. Extracting the relevant matrix entries, we have

$$\boldsymbol{\omega} = \begin{pmatrix}\widehat{\mathbf{z}} \cdot \dot{\widehat{\mathbf{y}}}\\ \widehat{\mathbf{x}} \cdot \dot{\widehat{\mathbf{z}}}\\ \widehat{\mathbf{y}} \cdot \dot{\widehat{\mathbf{x}}}\end{pmatrix}. \tag{14}$$

From (13, 14), we obtain $\mathcal{R}$'s kinetic energy T as a function of $\mathbf{q}$ and $\dot{\mathbf{q}}$:

$$T = \tfrac{1}{2} m \dot{\mathsf{M}}^2 + \tfrac{1}{2} \boldsymbol{\omega}^T \mathbf{I} \boldsymbol{\omega}, \tag{15}$$

where m is $\mathcal{R}$'s mass, and $\mathbf{I}$ is the moment of inertia matrix about axes through the centroid parallel to the local frame axes.

Example 3. Consider a Free Top: a general rigid body $\mathcal{R}$ moving under no forces, except the constraint that its centroid is fixed at the WF origin O. Define the local frame by O and two orthogonal unit vectors $\mathbf{u}, \mathbf{v}$ fixed in $\mathcal{R}$. Let $\mathbf{I}$ be the inertia matrix in this frame. In the working leading to (11), $\widehat{\mathbf{x}}_t$ and $\widehat{\mathbf{y}}_t$ equal $\mathbf{u}_t$ and $\mathbf{v}_t$ so we use the latter, whence $Q_t = [\mathbf{u}_t, \mathbf{v}_t, \mathbf{w}_t]$ where $\mathbf{w}_t = \mathbf{u}_t \times \mathbf{v}_t$. Let $\mathbf{q} = (\mathbf{u}, \mathbf{v})$.

There is no potential energy so the Lagrangian is just the kinetic energy, which is purely rotational. Thus the complete formulation, in terms of $\mathbf{q}$ and $\dot{\mathbf{q}}$, is

$$\mathbf{w} = \mathbf{u} \times \mathbf{v}, \quad \dot{\mathbf{w}} = \dot{\mathbf{u}} \times \mathbf{v} + \mathbf{u} \times \dot{\mathbf{v}}, \quad \boldsymbol{\omega} = \begin{pmatrix} \mathbf{w} \cdot \dot{\mathbf{v}} \\ \mathbf{u} \cdot \dot{\mathbf{w}} \\ \mathbf{v} \cdot \dot{\mathbf{u}} \end{pmatrix}, \quad \mathcal{L} = \tfrac{1}{2} \boldsymbol{\omega}^T \mathbf{I} \boldsymbol{\omega}, \quad \mathbf{C} = \begin{pmatrix} \mathbf{u} \cdot \mathbf{u} - 1 \\ \mathbf{v} \cdot \mathbf{v} - 1 \\ \mathbf{u} \cdot \mathbf{v} \end{pmatrix},$$

where the $_t$ have been dropped, $\mathcal{L}$ is the Lagrangian, and $\mathbf{C}$ is the vector of (residuals of) constraints. See the example code for this in §3 *Lagrangian facility.* □

Kinematic constraints. As with JBNCs, we make a pin joint in 2D and a spheric joint in 3D happen implicitly, without generating constraint equations, just by sharing a point between two bodies. Similarly, a revolute joint in 3D can happen implicitly by sharing a point on, and a vector along, the joint spine.

We try to use linear algebra ideas valid in any number of dimensions. E.g. in 3D, to say line AB is in the direction of $\mathbf{u}$, do not use cross product $\mathbf{u} \times (\mathsf{B} - \mathsf{A}) = \mathbf{0}$, which makes three equations but only two are independent. Instead say $\mathsf{B} - \mathsf{A} = \mu \mathbf{u}$: 3 scalar equations plus a new scalar variable μ, correctly removing $3 - 1 = 2$ DOF. Usually μ must be put in $\mathbf{q}$. If this is part of the definition of a cylindric or prismatic joint, μ might have the practical meaning of an actuator position.

One may reduce the number of coordinates by using *assignments.* E.g. the cylindrical joint in §4.2 is modelled in $\mathsf{B} - \mathsf{A} = \mu \mathbf{u}$ form, with A on one body, B on another, and $\mathbf{u}$ shared between the bodies. But by the assignment $\mathsf{B} := \mathsf{A} + \mu \mathbf{u}$, we keep B out of $\mathbf{q}$ and make the joint happen implicitly. Doing this depends on how $\mathbf{q}$'s components relate to the joint and is not always possible.

Shared vectors and points. One of the main uses of *vectors* is to be shared between parts, e.g. the definition of the revolute and cylindric joints in the RSCR mechanism of §4.2 involves the joined parts having a common vector. We use vectors to give *direction*, not distance. Any positive multiple has the same meaning: in effect, vector $\mathbf{v}$ denotes the unit vector $\mathbf{v}/|\mathbf{v}|$. JBNCs (see [4, §1.2.2]) also use this policy, that for vectors, direction is relevant but length isn't.

Shared *points* give no difficulty when two bodies have at most one in common. But as mentioned in §2.1, with two shared points they do, so we forbid this. The desired effect is gained by sharing a point and a vector.

2.3 The 2D case

This is similar to 3D but with smaller matrices:

- There are two BPVs O, A, where O must be a point.

- The local frame is the one in which $[\mathsf{O},\mathsf{A}] = \begin{bmatrix} 0 & r_{11} \\ 0 & 0 \end{bmatrix}$, then $R = [r_{11}]$ is 1×1.
- (6) becomes $S = \begin{bmatrix} -\alpha \\ 1 \end{bmatrix}$, with α as there.
- $\otimes$ acts on a 2×1 matrix $\begin{bmatrix} u \\ v \end{bmatrix}$ to produce the 2×2 matrix $\begin{bmatrix} u \\ v \end{bmatrix} \otimes = \begin{bmatrix} u & -v \\ v & u \end{bmatrix}$.
- $U = SR^{-1}$ as before, and (11) becomes $[\widehat{\mathbf{x}}, \widehat{\mathbf{y}}] = Q = \big([\mathsf{O},\mathsf{A}]\; U\big) \otimes$.
- $Q^T \dot{Q} = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix}$ with scalar angular velocity ω, and (14, 15) become

$$\omega = \widehat{\mathbf{y}} \cdot \dot{\widehat{\mathbf{x}}}, \qquad T = \tfrac{1}{2} m \dot{\mathsf{M}}^2 + \tfrac{1}{2} I\, \omega^2,$$

where I is the scalar moment of inertia about the centroid.

3 The numerical infrastructure

The DAETS solver. The DAETS (DAE by Taylor Series) solver [11, 12] accepts a system of n DAEs in n state variables $x_j(t)$, $1 \le j \le n$:

$$f_i(\, t, \text{ the } x_j \text{ and derivatives of them}\,) = 0, \quad 1 \le i \le n. \tag{16}$$

The f_i can be arbitrary expressions built from the x_j and t using arithmetic operations, standard functions (sin, exp, etc.), and the $\mathrm{d}^p/\mathrm{d}t^p$ operation. We assume the f_i are sufficiently differentiable and hence exclude functions such as abs, min, and max in their definitions.

This solver implements a variable-stepsize, fixed-order[2], explicit Taylor series (TS) method, where a typical order is in the range 12–20. Since the underlying method for computing TS is not affected by the index, DAETS handles *any index* DAE. Furthermore, ODEs and pure algebraic systems are handled as a particular case of the general formulation (16).

We outline how TS are computed for the simple pendulum given as a second-order, index-3 DAE:

$$\begin{aligned} 0 &= \ddot{u} + \lambda u \\ 0 &= \ddot{v} + \lambda v - G \\ 0 &= u^2 + v^2 - L. \end{aligned}$$

Here $\big(u(t), v(t)\big)$ is position, $\lambda(t)$ is a Lagrange multiplier, G is gravity, and L is the length of the pendulum.

[2]An order is chosen at the beginning of an integration and is fixed throughout it.

Write (at $t = 0$ without loss) $u(t) = u_0 + u_1 t + u_2 t^2 + \cdots$ and similarly for $v(t)$ and $\lambda(t)$. Substituting them into

$$\begin{aligned} a(t) &= \ddot{u} + \lambda u && (17)\\ b(t) &= \ddot{v} + \lambda v - G && (18)\\ c(t) &= u^2 + v^2 - L, && (19) \end{aligned}$$

we have for the coefficients in the expansions of $a(t)$, $b(t)$, and $c(t)$:

$$\begin{cases} a_0 = 2u_2 + \lambda_0 u_0 \\ b_0 = 2v_2 + \lambda_0 v_0 - G \\ c_0 = u_0^2 + v_0^2 - L^2 \end{cases} \qquad (20)$$

$$\begin{cases} a_1 = 6u_3 + \lambda_0 u_1 + \lambda_1 u_0 \\ b_1 = 6v_3 + \lambda_0 v_1 + \lambda_1 v_0 \\ c_1 = 2u_0 u_1 + 2v_0 v_1 \end{cases} \qquad (21)$$

$$\begin{cases} a_2 = 24u_4 + \lambda_0 u_2 + \lambda_1 u_1 + \lambda_2 u_0 \\ b_2 = 24v_4 + \lambda_0 v_2 + \lambda_1 v_1 + \lambda_2 v_0 \\ c_2 = 2u_0 u_2 + u_1^2 + 2v_0 v_2 + v_1^2 \end{cases} \qquad (22)$$

and so on. We refer to the above coefficients as Taylor coefficients (TCs). By equating the TCs in (20, 21, 22) to zero, we solve for the TCs of u, v, λ. This is not obvious from (20, 21, 22), but proceeds as follows.

Given initial values for u_0 and v_0, check if they satisfy the constraint

$$0 = c_0 = u_0^2 + v_0^2 - L^2, \qquad (23)$$

and if not adjust u_0 or v_0 or both.

Using u_0, v_0 as constants, and given values for u_1 and v_1, check if they satisfy the constraint

$$0 = c_1 = 2(u_0 u_1 + v_0 v_1), \qquad (24)$$

and if not adjust u_1 or v_1 or both.

Then for $k = 0$ up to some order K, we solve a linear system

$$0 = a_{k+0}, b_{k+0}, c_{k+2} \quad \text{for} \quad u_{k+2}, v_{k+2}, \lambda_{k+0} \qquad (25)$$

using previously computed coefficients as constants. E.g., when $k = 0$, we solve

$$\left.\begin{array}{l} 0{=}a_0{=}2u_2{+}\lambda_0 u_0 \\ 0{=}b_0{=}2v_2{+}\lambda_0 v_0{-}G \\ 0{=}c_2{=}2u_0u_2{+}2v_0v_2{+}u_1^2{+}v_1^2 \end{array}\right\}, \text{ i.e. } \begin{bmatrix} 2 & 0 & u_0 \\ 0 & 2 & v_0 \\ 2u_0 & 2v_0 & 0 \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \\ \lambda_0 \end{bmatrix} = \begin{bmatrix} 0 \\ G \\ -u_1^2 - v_1^2 \end{bmatrix} \qquad (26)$$

for u_2, v_2, λ_0 using u_0, v_0, u_1, v_1 as known.

The computational scheme for TCs is guided by two nonnegative integer vectors, *equation offsets* c_i and *variable offsets* d_j, here $(0, 0, 2)$ and $(2, 2, 0)$, respectively;

cf. (25). These vectors are found by Pryce's structural analysis (SA) [18], details omitted. There is a set of "SA-amenable" DAEs that (when smooth enough) can be expanded in TS in a similar way. This is the same set on which one can use Pantelides SA [17] and the Mattsson–Söderlind dummy derivative method [10]. It includes all index-3 Euler-Lagrange DAEs [18, Theorem 5.3].

From

$$u(t) \approx \widetilde{u}(t) = \sum_{i=0}^{K+2} u_i t^i, \quad v(t) \approx \widetilde{v}(t) = \sum_{i=0}^{K+2} v_i t^i, \quad \lambda(t) \approx \widetilde{\lambda}(t) = \sum_{i=0}^{K} \lambda_i t^i,$$

given stepsize h, we compute an approximate TS solution at $t_1 = h$, and by differentiating $\widetilde{u}(t)$ and $\widetilde{v}(t)$, we approximate $\dot{u}$ and $\dot{v}$ at t_1 and repeat the above process starting at t_1.

We present a few implementation details below. DAETS requires a templated `C++` function for evaluating the f_i's. The simple pendulum is encoded e.g. as

```
template <typename T>
void fcn(T t, const T* x, T* f, void* param) {
   double G = 9.81, L = 10.0;
   T u = x[0], v = x[1], lam = x[2];
   f[0] = Diff(u, 2) + lam * u;
   f[1] = Diff(v, 2) + lam * v - G;
   f[2] = sqr(u) + sqr(v) - sqr(L);
}
```

On input, `t` is the time variable (not used here), `x[j-1]` contains the jth state variable; on output `f[i-1]` stores f_i. The last argument, `param` (not used here), is to pass parameters from the solver. The `Diff`$(\cdot, p)$ function implements $\mathrm{d}^p/\mathrm{d}t^p$.

Before integration starts, DAETS performs SA by executing the f_i's code, here `fcn`, through operator overloading to find the DAE index, DOF, and the offset vectors. Then the solver executes the same function with FADBAD++'s Taylor series class to build in memory a computational graph, which is evaluated on each integration step to compute TCs for the f_i's; in our example (20, 21, 22).

The constraints of the problem are satisfied by orthogonal projection. For example, denoting $\mathbf{u} = (u, v)$, $\mathbf{u}_0 = (u_0, v_0)$, (23) is done by solving

$$\min_{\mathbf{u}} \|\mathbf{u} - \mathbf{u}_0\|_2 \quad \text{s.t.} \quad 0 = u^2 + v^2 - L^2;$$

similarly for (24).

Since the stepsize is selected based on a tolerance specified by the user, the TS solution on each integration step is very close to satisfying the constraints, and one or two iterations of Gauss-Newton suffice to project this solution. On the first step though, the IVs given by the user may not be close to satisfying the constraints: here we use the IPOPT [23] optimisation package, see below. In terms of IVs, the user can fix one of the values for u and v, and DAETS would try to solve for the other; similarly for $\dot{u}$, $\dot{v}$.

Computing TCs requires solving linear systems, all with the same matrix (which is factorised when $k = 0$), the Jacobian $\mathbf{J} = (J_{ij})$ determined by the offsets c_i, d_j:

$$J_{ij} = \partial f_i / \partial x_j^{(d_j - c_i)}, \quad \text{or 0 where } d_j < c_i, \tag{27}$$

for details see [11]. We handle sparsity using the Klu solver [2].

Lagrangian facility. Given a `C++` encoding of a Lagrangian function $\mathcal{L}$ and expressions for the C_i in (2), this facility applies the backward/reverse mode of AD to perform the partial differentiations in (1) and applies `Diff`$(\cdot, 1)$ to the result of $\frac{\partial \mathcal{L}}{\partial \dot{q}_j}$, and together with (2), constructs a system in the form of (16) of $n_q + n_c$ equations f_i in variables $(q_1, \ldots, q_{n_q}, \lambda_1, \ldots, \lambda_{n_c})$.

Below on the left are the Lagrangian for the simple pendulum and the length constraint, and on the right is the Daets function using this facility:

$$\mathcal{L} = \tfrac{1}{2}(\dot{u}^2 + \dot{v}^2) + Gv \tag{28}$$

$$C_1 = u^2 + v^2 - L^2 \tag{29}$$

```
 1  template <typename T>
 2  void PEND(T t, const T* x, T* f, void* param) {
 3    double G = 9.81, L = 10.0;
 4    int nq = 2, nc = 1;
 5    std::vector<fadbad::B<T>> q(nq), qdot(nq), C(nc);
 6    init_q_qp(x, q, qdot);
 7    B<T> Lag;
 8    {
 9      Lag = 0.5*(sqr(qdot[0])+sqr(qdot[1])) + G*q[1];
10      C[0] = sqr(q[0]) + sqr(q[1]) - sqr(L);
11    }
12    setupEquations(Lag, x, q, qdot, C, f);
13  }
```

The block between lines 8 and 11 is where a Lagrangian and constraints are provided; the rest is boilerplate.

The function `init_q_qp` "connects" the input variables at `x` to the $\mathbf{q} = (u, v)$ and $\dot{\mathbf{q}} = (\dot{u}, \dot{v})$ vectors, whose components are of templated backward differentiation type `B<T>` of FADBAD++. The `setupEquations` function takes as input the $\mathcal{L}$ and C_1, stored in `Lag` and `C[0]`, respectively, and `x`, `q`, `qdot`; performs the differentiations in (1) using FADBAD++, produces from (28, 29) a computational graph in memory to evaluate[3] the TCs of (17, 18, 19), and stores the result at the output parameter `f`. From the solver's perspective, `fcn` and `FCN` perform the same computation. The main advantage of the latter is that we do not need to derive and program the actual differentiated equations: they are constructed automatically at runtime, and invisible to the user.

To improve the performance of the AD in FADBAD++, we have implemented techniques to eliminate common subexpressions when building a computational graph [9]. In effect symbolic manipulation is being done, but entirely behind the scenes and without symbolic algebra software.

As another example of the Lagrangian facility, we show below the encoding of the Free Top from Example 3. For brevity it is assumed that the axes are the

[3]In practice $a(t)$ and $b(t)$ in (17, 18) become $a(t) = \ddot{u} + 2u\lambda$ and $b(t) = \ddot{v} + 2\lambda v - G$, respectively, but this does not affect the computation.

principal axes of inertia so that only the diagonal entries I_u, I_v, I_w of the inertia matrix $\mathbf{I}$ are nonzero. We omit initial code that copies from arrays holding $\mathbf{q}, \dot{\mathbf{q}}$ to variables `u,v,udot,vdot` holding $\mathbf{u}, \mathbf{v}, \dot{\mathbf{u}}, \dot{\mathbf{v}}$. These variables are of a 3-vector class `vec3` that supports linear operations like `+`, and dot and cross product written as `*` and `cross` respectively.

$$\begin{aligned}
&\mathbf{w} = \mathbf{u}\times\mathbf{v}\\
&\dot{\mathbf{w}} = \dot{\mathbf{u}}\times\mathbf{v} + \mathbf{u}\times\dot{\mathbf{v}}\\
&\omega_u=\mathbf{w}\cdot\dot{\mathbf{v}},\ \omega_v=\mathbf{u}\cdot\dot{\mathbf{w}},\ \omega_w=\mathbf{v}\cdot\dot{\mathbf{u}}\\
&0 = C_1 = \mathbf{u}\cdot\mathbf{u} - 1\\
&0 = C_2 = \mathbf{v}\cdot\mathbf{v} - 1\\
&0 = C_3 = \mathbf{u}\cdot\mathbf{v}\\
&\mathcal{L} = \tfrac{1}{2}(I_u\omega_u^2 + I_v\omega_v^2 + I_w\omega_w^2)
\end{aligned}$$

```
vec3 w = cross(u,v),
     wp = cross(udot,v) + cross(u,vdot);
B<T> omu = w*vdot, omv = u*wp, omw = v*udot;
C[0] = u*u - 1;
C[1] = v*v - 1;
C[2] = u*v;
Lag=0.5*(Iu*sqr(omu)+Iv*sqr(omv)+Iw*sqr(omw));
```

Finally, we have also constructed and experimented with a Hamiltonian facility, which allows one to provide a Hamiltonian function, and the resulting equations are produced through AD in a similar manner.

Mechanism facility. This lets one specify a mechanism by a *MechSpec* text file, which the facility parses and converts to a Lagrangian plus constraints. That is, a generic function similar to `FCN` determines a Lagrangian and constraints from a MechSpec file. It is written in YAML, a human-readable data serialisation language. YAML readers exist for `C++` and other programming languages to convert the file to an internal program data structure. We say little about the syntax and semantics of YAML and MechSpec—for details see [14]—and hope the examples used are self-explanatory.

IPOPT. DAETS incorporates the widely used IPOPT software library [23] for large scale nonlinear optimisation. DAETS uses it to find an initial "consistent point" of a general DAE. In the mechanism context this is the *initial position problem*: given DOF (initial) constraining equations on $\mathbf{q}$, e.g. the values of DOF components, to fix the whole of $\mathbf{q}$; and similarly for $\dot{\mathbf{q}}$.

The $\mathbf{q}$ problem is often highly nonlinear with multiple solutions, so guesses of some components are necessary to ensure convergence to the desired initial position. Success depends on the quality of the guesses one can supply, but subject to this our experience is that IPOPT is a robust solver of this problem. The $\dot{\mathbf{q}}$ problem by contrast is always linear.

4 Examples

We illustrate our approach on one 2D and two 3D examples in §4.1 to §4.3.

4.1 Mechanism0

4.1.1 Specification, Lagrangian and constraints

Figure 3 shows on the right a picture of Mechanism0: rod OA pivots at fixed point O; at its end pivots a spring AB; a particle is attached at B. The system

```
1  Title: Mechanism0
2  PhysicalParams: {L: 2, M: 1,
3    k: 1000, l: 1, mu: .5, m: .1}
4  Dimension: 2
5  PartData:
6    Fixed: {O: [0,0]}
7    Rigids:
8      OA: {Geom: [[L,0]], Dyna: [[L/2,0],M,M*L**2/12]}
9    Springs:
10     AB: [ k, l, mu ]
11   Particles:
12     B: m
13 AppliedForces:
14   Gravity:
15 ProblemData:
16   t0: 0
17   tend: 1
18   positions: {A: [0,-L], B: [0,-(L+l)]}
19   fixedpositions: {Ax, B}
20   velocities: {Ax: 1, B: [0,0]}
```

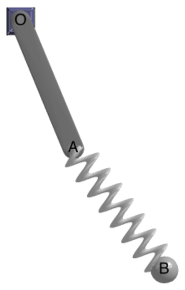


Figure 3: Mechanism0 specification in YAML.

moves under gravity. On the left is a MechSpec description of it (lines 1–14) and a particular IVP for numerical solution (lines 15–20).

Line 8 specifies the geometry and dynamics of rod OA. In **Geom**, the coordinates of the first point are not stored as always $(0, 0)$, and `[L,0]` gives $(\mathsf{A}_x, \mathsf{A}_y) = (L, 0)$. In **Dyna**, `[L/2,0]` is the centre of mass, `M` is the mass, and `M*L**2/12` is the moment of inertia. Line 10 defines AB to be a spring of stiffness k, rest length ℓ and mass μ (modelled as a stretchable uniform rod). Line 12 defines B to be a particle of mass m. Line 14 makes the system move under default gravity. Lines 2–3 give values to the various parameters (in SI units since this is what is used for default gravity).

Lines 15–20 define a numerical IVP: integrate over $0 \leq t \leq 1$ with initially the system hanging vertically with the spring at its rest length, and point A given $1\,\mathrm{ms}^{-1}$ velocity in the x direction. Line 19 specifies that the initial positions of A_x, B_x, and B_y must not be changed when finding a consistent initial point.

From the MechSpec description, the mechanism facility defines the coordinate vector (A_x is the x-coordinate of A, and so on)

$$\mathbf{q} = (\mathsf{A}, \mathsf{B}) = (\mathsf{A}_x, \mathsf{A}_y, \mathsf{B}_x, \mathsf{B}_y).$$

and constructs all the kinetic and potential energy contributions:

$$\left.\begin{array}{lll}
\text{KE} & \text{gravitational PE} & \text{spring PE} \\
T_{\mathsf{OA}} = \dfrac{M}{6}|\dot{\mathsf{A}}|^2, & V_{\mathsf{OA}} = \dfrac{Mg}{2}\mathsf{A}_y, & \\
T_{\mathsf{AB}} = \dfrac{\mu}{6}(|\dot{\mathsf{A}}|^2+\dot{\mathsf{A}}\cdot\dot{\mathsf{B}}+|\dot{\mathsf{B}}|^2), & V^{[g]}_{\mathsf{AB}} = \dfrac{\mu g}{2}(\mathsf{A}_y+\mathsf{B}_y), & V^{[s]}_{\mathsf{AB}} = \dfrac{k}{2}\left(|\mathsf{B}-\mathsf{A}|-\ell\right)^2, \\
T_{\mathsf{B}} = \dfrac{m}{2}|\dot{\mathsf{B}}|^2, & V_B = mg\,\mathsf{B}_y. &
\end{array}\right\} \qquad (30)$$

Here, the expression T_{AB} is the KE (in any number of dimensions) of a thin rod of mass μ with endpoints A, B.

Then the system is specified by the Lagrangian, plus the rod's length constraint:

$$\mathcal{L} = \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = T - V = (T_{\mathsf{OA}} + T_{\mathsf{AB}} + T_{\mathsf{B}}) - (V_{\mathsf{OA}} + V^{[g]}_{\mathsf{AB}} + V^{[s]}_{\mathsf{AB}} + V_{\mathsf{B}}), \qquad (31)$$

$$0 = l_{\mathsf{OA}}(\mathbf{q}) = |\mathsf{A}|^2 - L^2. \qquad (32)$$

These are processed by the Lagrangian facility.

Including angles. An angle θ between two bodies is needed for instance to measure the position of a rotational actuator, or the work done by a torque. A $\mathbf{q}$ made of only cartesian coordinates cannot do this safely—it thinks θ and $\theta + 2\pi$ are the same—so we include θ explicitly in $\mathbf{q}$. We show this here just for the 2D case.

Suppose Mechanism0 has a constant torque τ on rod OA—that is, between OA and the WF. We omit how this appears in the data file, but the effect on the model is as follows. (a) Append to $\mathbf{q}$ a variable θ_{OA} defined as the angle from the WF x axis to OA's local-frame x axis. (b) Replace (32) by two equations $\mathsf{A}_x = L\cos\theta_{\mathsf{OA}}$, $\mathsf{A}_y = L\sin\theta_{\mathsf{OA}}$. (c) Include in (30), hence (31), the PE contribution

$$V^{[\tau]}_{\mathsf{OA}} = -\tau\,\theta_{\mathsf{OA}}. \qquad (33)$$

The Euler–Lagrange DAE then automatically tracks θ_{OA} as a continuous quantity.

4.1.2 Verification and numerical results for Mechanism0

As a check, we described Mechanism0 in a Lagrangian form without constraints, leading to an ODE, transcribed into MATLAB and solved using its ODE suite. This and the mechanism facility solution agree closely; we take this as sufficient evidence that both models and implementations are correct.

The chosen coordinates are

$$\mathbf{q} = (\theta, r, \phi)$$

with

θ : angle of OA from downward vertical;
r : current length of spring AB;
ϕ : angle of AB from downward vertical.

With these coordinates, we have positions and velocities

$$\left.\begin{array}{ll} \mathsf{A} = L\,(\sin\theta, -\cos\theta), & \mathsf{B} = \mathsf{A} + r\,(\sin\phi, -\cos\phi), \\ \dot{\mathsf{A}} = L\,(\cos\theta, \sin\theta)\dot\theta, & \dot{\mathsf{B}} = \dot{\mathsf{A}} + (\sin\phi, -\cos\phi)\dot r + r\,(\cos\phi, \sin\phi)\dot\phi. \end{array}\right\}$$

The Lagrangian is still given by (31) with extra term (33). Now (30) becomes

$$\left.\begin{array}{lll} \text{KE} & \text{gravitational PE} & \text{spring or torque PE} \\ T_{\mathsf{OA}} = \tfrac16 ML^2\dot\theta^2, & V_{\mathsf{OA}} = -\tfrac12 MgL\cos\theta & V^{[\tau]}_{\mathsf{OA}} = -\tau\theta \\ \multicolumn{3}{l}{T_{\mathsf{AB}} = \tfrac12\mu\Big(L^2\dot\theta^2 + L\sin(\phi-\theta)\dot\theta\dot r + Lr\cos(\phi-\theta)\dot\theta\dot\phi + \tfrac13(\dot r^2 + r^2\dot\phi^2)\Big),} \\ & V^{[g]}_{\mathsf{AB}} = -\mu g(L\cos\theta + \tfrac12 r\cos\phi), & V^{[s]}_{\mathsf{AB}} = \tfrac12 k\,(r-\ell)^2\,, \\ \multicolumn{3}{l}{T_{\mathsf{B}} = \tfrac12 m\Big(L^2\dot\theta^2 + 2L\sin(\phi-\theta)\dot\theta\dot r + 2Lr\cos(\phi-\theta)\dot\theta\dot\phi + \dot r^2 + r^2\dot\phi^2\Big),} \\ & & V_{\mathsf{B}} = -mg(L\cos\theta + r\cos\phi), \end{array}\right\}$$

where θ_{OA} has been changed to θ in the torque PE, which is valid because these two angles differ by a constant. This gives three equations of motion

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\mathbf{q}}} = \frac{\partial\mathcal{L}}{\partial\mathbf{q}}, \quad \text{call the latter } F(\mathbf{q}, \dot{\mathbf{q}}). \tag{34}$$

For the numerics, introduce the Hamiltonian generalised momentum vector

$$\mathbf{p} = \partial\mathcal{L}/\partial\dot{\mathbf{q}}, \qquad \text{here, } \dot{\mathbf{q}} = (\dot\theta, \dot r, \dot\phi).$$

Now $\mathbf{p} = \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}$—linear in $\dot{\mathbf{q}}$—where $\mathbf{M}$ is the symmetric mass matrix. Here

$$\mathbf{M} = \mathbf{M}(\theta, r, \phi) = \\ \begin{bmatrix} \left(\tfrac13 M + \mu + m\right)L^2 & (\tfrac12\mu + m)L\sin(\phi-\theta) & (\tfrac12\mu + m)Lr\cos(\phi-\theta) \\ (\tfrac12\mu + m)L\sin(\phi-\theta) & \tfrac13\mu + m & 0 \\ (\tfrac12\mu + m)Lr\cos(\phi-\theta) & 0 & (\tfrac13\mu + m)r^2 \end{bmatrix}$$

Now (34) becomes two first-order vector ODEs, making 6 scalar equations:

$$\dot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}\mathbf{p}, \qquad \dot{\mathbf{p}} = F(\mathbf{q}, \dot{\mathbf{q}}).$$

Note the block triangular structure: find $\dot{\mathbf{q}}$ first, and use it to find $\dot{\mathbf{p}}$.

The mechanism facility program and the MATLAB program read the same YAML problem file. The IVs specify the position and velocity of $\mathsf{A} = (\mathsf{A}_x, \mathsf{A}_y)$ and $\mathsf{B} = (\mathsf{B}_x, \mathsf{B}_y)$. They might not be consistent: A might not lie on, and $\dot{\mathsf{A}}$ might not be tangential to, the circle of radius L round O. DAETS finds consistent points by calling the optimisation code IPOPT. To make the result predictable, we use DAETS's ability to specify some IVs "fixed", not to be modified by IPOPT. Here, one of A_x, A_y may be fixed and also one of $\dot{\mathsf{A}}_x, \dot{\mathsf{A}}_y$. To ensure both codes solve the same IVP, the MATLAB code is made to do the same, e.g. if A_x is "fixed" it sets

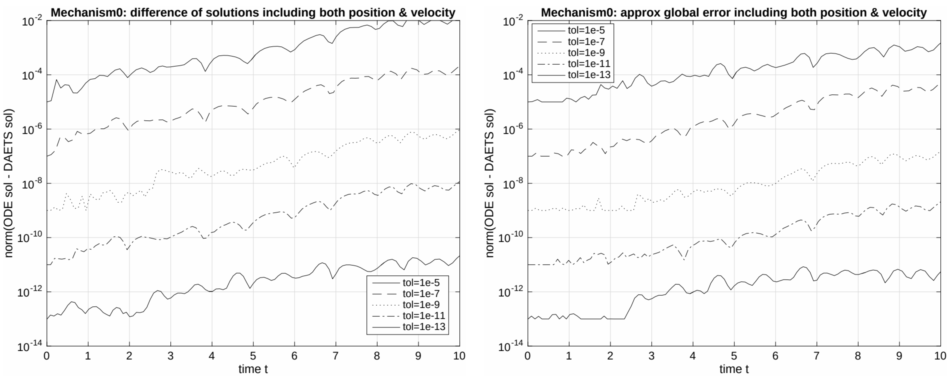


Figure 4: (a) Comparison of DAETS and `ode113` solutions at different tolerances. (b) Estimated global error: (`ode113` solution at `tol` = 2.2e-14) – (DAETS solution).

$\mathsf{A}_y = \pm\sqrt{L^2 - A_x^2}$. We omit details to do with choice of sign and possible numerical ill-conditioning.

To test agreement between DAETS and ODE solution, we used `ode113`. For each tolerance `tol`, the 8-element vector $\mathbf{v} = (\mathsf{A}, \mathsf{B}, \dot{\mathsf{A}}, \dot{\mathsf{B}})$ (positions and velocities) was formed for each solver at each t. A feature of the MATLAB ODE suite is that output can be produced, to full accuracy, at each t in a vector of values that is given as input to the solver call. That is, we produced output with `ode113` at points t selected by DAETS.

Figure 4(a) shows $\|\mathbf{v}_{\text{DAETS}} - \mathbf{v}_{\text{ODE}}\|_2$ against t for 5 tolerances `tol`, with a vertical log scale. The growth of the difference is much the same for each `tol`: about 3 orders of magnitude over the range. The curves at different `tol` obviously are strongly correlated; so far we have not looked into why this is.

Figure 4(a) shows an assessment of the global error of the DAETS solution. Namely $\mathbf{v}_{\text{ODE}}$ (at the given `tol`) is replaced by a reference solution $\mathbf{v}_{\text{ref}}$ which we took to be the `ode113` solution with `tol` = 2.2e-14, about the smallest tolerance it allows. So it plots $\|\mathbf{v}_{\text{DAETS}} - \mathbf{v}_{\text{ref}}\|_2$ against t for each `tol`. The plots show these errors grow more slowly than the differences on the left: they are about ten times smaller at $t = 10$. This suggests the difference $\mathbf{v}_{\text{DAETS}} - \mathbf{v}_{\text{ODE}}$ is mostly due to global error in the `ode113` solutions, and DAETS is somewhat more accurate than `ode113`.

4.2 The RSCR mechanism

Specification and design parameters. Figure 5 shows on the right a picture of the RSCR (Revolute-Spherical-Cylindrical-Revolute) mechanism, a four-bar linkage in 3D with 1 DOF, of which the classic plane four-bar linkage is a special case. On the left is a MechSpec description of it.

```
Dimension: 3
PartData:
  Fixed:
    D: [0,0,0]
    A: [-L0,0,0]
    w: [cos(d2r*d),sin(d2r*d),0]
    u: [cos(d2r*a)*sin(d2r*b),
        sin(d2r*a)*sin(d2r*b),cos(d2r*b)]
  Rigids:
    ABu: {Geom:[[L1],[0,1]],
          Dyna:[[L1/2],m1,m1*L1**2/12]}
    CBv: {Geom:[[L2],[0,1]],
          Dyna:[[L2/2],m2,m2*L2**2/12]}
    DC1wv: {Geom:[[L3],[0,1],[0,cos(c),sin(c)]],
            Dyna:[[L3/2],m3,m3*L3**2/12]}
  Collinears: {C1vC}
```

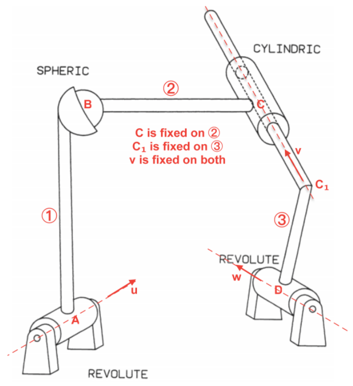


Figure 5: RSCR mechanism. MechSpec text on the left. The 8 parameters defining the geometry are L_0, L_1, L_2, L_3 (lengths $\underline{\mathsf{DA}}$, $\underline{\mathsf{A}}\mathsf{B}$, BC, $\mathsf{C_1}\underline{\mathsf{D}}$), and $\alpha, \beta, \gamma, \delta$ (angles, see text), written `a,b,c,d` in the MechSpec; `d2r` stores $\pi/180$. By a "padding convention", omitted trailing coordinates are zero, e.g. `[L1]` means `[L1,0,0]`, the point $(L_1, 0, 0)$.

As a mnemonic, WF-fixed items are underlined, e.g. $\underline{\mathsf{A}}$ for a point, $\underline{\mathbf{u}}$ for a vector; others are moving. The moving parts are ①, ②, ③, with joints at $\underline{\mathsf{A}}$, B, C, $\underline{\mathsf{D}}$. The joint spines at $\underline{\mathsf{A}}$ and $\underline{\mathsf{D}}$ are fixed in the WF, along vectors $\underline{\mathbf{u}}$ and $\underline{\mathbf{w}}$ respectively. Point $\mathsf{C_1}$ is fixed on ③ and on the cylindric joint spine which is along $\mathbf{v}$. We can put $\underline{\mathsf{A}}, \mathsf{C}, \mathsf{C_1}, \underline{\mathsf{D}}$ anywhere along the relevant spines, so without loss assume

- line $\underline{\mathsf{A}}\mathsf{B} \perp$ (perpendicular to) spine at $\underline{\mathsf{A}}$;
- line $\mathsf{CB} \perp$ spine at C;
- line $\underline{\mathsf{D}}\mathsf{C_1} \perp$ spines at both $\underline{\mathsf{D}}$ and $\mathsf{C_1}$.

We fix the WF to have origin at $\underline{\mathsf{D}}$ with $\underline{\mathsf{DA}}$ the negative x axis, $\underline{\mathsf{A}}$ at $(-L_0, 0, 0)$, and the joint spine at $\underline{\mathsf{D}}$ in the xy plane. The latter's direction $\underline{\mathbf{w}}$ is arbitrary in this plane, say $\underline{\mathbf{w}} = (\cos\delta, \sin\delta, 0)^T$. The spine at $\underline{\mathsf{A}}$ is now in an arbitrary WF direction, say $\underline{\mathbf{u}} = (\cos\alpha\sin\beta, \sin\alpha\sin\beta, \cos\beta)^T$ with spherical polar angless α, β.

In ③, the cylindric and the revolute joint spines are skew, non-intersecting lines in general. Projected on a plane normal to $\underline{\mathsf{D}}\mathsf{C_1}$, the former, at $\mathsf{C_1}$, is rotated an arbitrary γ from the latter, at $\underline{\mathsf{D}}$, so in ③'s local frame it is along

$$\mathbf{v} = (0, \cos\gamma, \sin\gamma)^T \tag{35}$$

Vector $\mathbf{v}$ moves in the WF and is fixed in both ② and ③.

Local frames and coordinate vector. We choose ①'s local origin at $\underline{\mathsf{A}}$, ②'s at C, ③'s at $\underline{\mathsf{D}}$, and in each case the x axis along the part's length, and the xy plane defined by a unit vector along the joint spine that passes through the local origin. Since these spines are orthogonal to the part lengths by construction, the R matrix (5) of each frame is diagonal.

Table 1: RSCR: assignments and equations.

Bring in	**Assign**	**Equate**	**DOF**	
③: $\underline{\mathsf{D}}\mathsf{C}_1\underline{\mathbf{w}}\mathbf{v}$ rigidity.				
C_1	$\widehat{\mathbf{x}}_3 := \mathsf{C}_1/L_3$ $\widehat{\mathbf{z}}_3 := \widehat{\mathbf{x}}_3\times\underline{\mathbf{w}}$ $\mathbf{v} := \cos(\gamma)\underline{\mathbf{w}} + \sin(\gamma)\widehat{\mathbf{z}}_3$	$0 = \mathsf{C}_1^2 - L_3^2$ $0 = \mathsf{C}_1 \cdot \underline{\mathbf{w}}$	$+3\ -2$	(37)
②: $\mathsf{CB}\mathbf{v}$ rigidity, and specify cylindric joint				
B, μ	$\mathsf{C} := \mathsf{C}_1 + \mu\mathbf{v}$	$0 = (\mathsf{B}-\mathsf{C})^2 - L_2^2$ $0 = (\mathsf{B}-\mathsf{C})\cdot\mathbf{v}$	$+3+1\ -2$	(38)
①: $\underline{\mathsf{A}}\mathsf{B}\underline{\mathbf{u}}$ rigidity				
		$0 = (\mathsf{B}-\underline{\mathsf{A}})^2 - L_1^2$ $0 = (\mathsf{B}-\underline{\mathsf{A}})\cdot\underline{\mathbf{u}}$	-2	(39)
		Net DOF:	$1 = +7\ -6$	

To specify the constraints in Table 1, only ③ needs full frame details, given by the BPVs $\underline{\mathsf{D}}\mathsf{C}_1\underline{\mathbf{w}}$ in that order, so from (5, 6, 8), its U-matrix U_3 comes from

$$\mathbf{A}_3{=}[\underline{\mathsf{D}}, \mathsf{C}_1, \underline{\mathbf{w}}]{=}\begin{bmatrix}0 & L_3 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{bmatrix}, \qquad U_3 = \begin{bmatrix}-1 & 0\\ 1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}L_3 & 0\\ 0 & 1\end{bmatrix}^{-1} = \begin{bmatrix}-\frac{1}{L_3} & 0\\ \frac{1}{L_3} & 0\\ 0 & 1\end{bmatrix}.$$

The moving basic points are C_1, B. Bringing in parts in order ③, ②, ① as in Table 1, we express $\mathbf{v}$ by an assignment in terms of C_1, then C by an assignment in terms of C_1, $\mathbf{v}$ and a scalar μ. So the coordinate vector need only be

$$\mathbf{q} = (\mathsf{C}_1, \mathsf{B}, \mu) \qquad \text{with } 3\times 2+1 = 7 \text{ scalar components.} \tag{36}$$

Kinematics: the constraint equations. Here all positions and vectors are in the WF. The joints, handled as in §2.2 *Kinematic constraints*, generate no equations, so the only constraints are rigidity equations. Namely for each part: (i) a line joining two points has specified length and (ii) a unit vector is orthogonal to this line. This makes 2 equations per part, a total of 6.

The moving frame of ③ is needed to define vector $\mathbf{v}$ along the spine of the cylindric joint. Denote its unit vectors (the columns of Q_3) as $\widehat{\mathbf{x}}_3, \widehat{\mathbf{y}}_3, \widehat{\mathbf{z}}_3$, where $\widehat{\mathbf{x}}_3$ along $\underline{\mathsf{D}}\mathsf{C}_1$ is defined by

$$\widehat{\mathbf{x}}_3 = (\mathsf{C}_1 - \underline{\mathsf{D}})/L_3 = \mathsf{C}_1/L_3. \tag{40}$$

Vector $\widehat{\mathbf{y}}_3$ is an alias of the fixed $\underline{\mathbf{w}}$, whose name will be used instead, and

$$\widehat{\mathbf{z}}_3 = \widehat{\mathbf{x}}_3 \times \underline{\mathbf{w}}.$$

From (35), in ③'s local frame

$$\mathbf{v} = (0, \cos(\gamma), \sin(\gamma))^T = \cos(\gamma)\underline{\mathbf{w}} + \sin(\gamma)\widehat{\mathbf{z}}_3. \tag{41}$$

Then (40, 41) remain true during motion, by (4). Finally, define the cylindric joint by saying C is on the line through C_1 along $\mathbf{v}$, i.e. for some scalar μ we have

$$\mathsf{C} = \mathsf{C}_1 + \mu\mathbf{v}. \tag{42}$$

Table 1 summarises this. Equations (40) to (42) become assignments that express $\widehat{\mathbf{x}}_3$, $\widehat{\mathbf{z}}_3$, $\mathbf{v}$ and C as functions of C_1, μ only. The "Bring in" column shows when elements of $\mathbf{q}$ enter the calculation. The "DOF" column is a running count—increased by "Bring in" items, not changed by "Assign" items, and reduced by "Equate" items.

Dynamics. Take a simple model where: ② is a fully 3D rigid body; ① and ③ have the simpler dynamics of (uniform) thin rods $\underline{\mathsf{A}}\mathsf{B}$, $\underline{\mathsf{D}}\mathsf{C}_1$ of masses m_1, m_3; the mass of joints is neglected; no forces act, so the Lagrangian $\mathcal{L}$ is just the total kinetic energy.

Part ②'s KE T_2 involves $\dot{\mathsf{C}}$, so differentiate the assignments in Table 1, and apply (11)–(15) to get ②'s moving frame, centroid velocity and angular velocity, shown in Table 2.

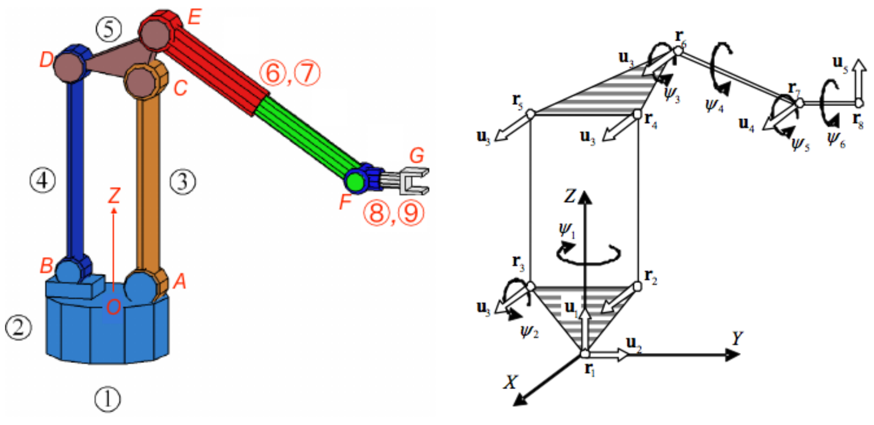
Similarly to the KE of rod $\mathsf{O}\underline{\mathsf{A}}$ in (30), the KE of ① is $T_1 = \frac{1}{6}m_1\dot{\mathsf{B}}^2$, and ③ is similar. Then $\mathcal{L}$ is given as a function of $\mathbf{q} = (\mathsf{C}_1, \mathsf{B}, \mu)$ and $\dot{\mathbf{q}} = (\dot{\mathsf{C}}_1, \dot{\mathsf{B}}, \dot{\mu})$ by

$$\mathcal{L} = T_1 + T_2 + T_3 = \left(\frac{m_1}{6}\dot{\mathsf{B}}^2\right) + \left(\tfrac{1}{2}m_2\dot{\mathsf{M}}_2^2 + \tfrac{1}{2}\boldsymbol{\omega}_2^T\,\mathbf{I}_2\,\boldsymbol{\omega}_2\right) + \left(\frac{m_3}{6}\dot{\mathsf{C}}_1^{\,2}\right),$$

see Table 2 for M_2. This, plus Table 1's constraints, gives the equations of motion. Fully 3D dynamics of parts ①, ③ would be handled as has been done for ②.

Table 2: RSCR dynamics calculation for simple model in text. $\overline{\mathsf{M}}_2$ is the fixed local position of body ②'s centroid (Mass centre), M_2 its moving WF position.

$\dot{\widehat{\mathbf{x}}}_3 := \dot{\mathsf{C}}_1/L_3,$	$\dot{\widehat{\mathbf{z}}}_3 := \dot{\widehat{\mathbf{x}}}_3 \times \underline{\mathbf{w}},$	$\dot{\mathbf{v}} := \sin(\gamma)\dot{\widehat{\mathbf{z}}}_3,$	$\dot{\mathsf{C}} := \dot{\mathsf{C}}_1 + \mu\dot{\mathbf{v}} + \dot{\mu}\mathbf{v},$
$\widehat{\mathbf{x}}_2 = (\mathsf{B} - \mathsf{C})/L_2,$	$\widehat{\mathbf{y}}_2 = \mathbf{v},$	$\widehat{\mathbf{z}}_2 = \widehat{\mathbf{x}}_2 \times \widehat{\mathbf{y}}_2,$	$Q_2 = [\widehat{\mathbf{x}}_2, \widehat{\mathbf{y}}_2, \widehat{\mathbf{z}}_2],$
$\dot{\widehat{\mathbf{x}}}_2 = (\dot{\mathsf{B}} - \dot{\mathsf{C}})/L_2,$	$\dot{\widehat{\mathbf{y}}}_2 = \dot{\mathbf{v}},$	$\dot{\widehat{\mathbf{z}}}_2 = \dot{\widehat{\mathbf{x}}}_2 \times \widehat{\mathbf{y}}_2 + \widehat{\mathbf{x}}_2 \times \dot{\widehat{\mathbf{y}}}_2,$	$\dot{Q}_2 = [\dot{\widehat{\mathbf{x}}}_2, \dot{\widehat{\mathbf{y}}}_2, \dot{\widehat{\mathbf{z}}}_2],$
$\dot{\mathsf{M}}_2 = \dot{\mathsf{C}} + \dot{Q}\overline{\mathsf{M}}_2,$		$\boldsymbol{\omega}_2 = \begin{pmatrix} \widehat{\mathbf{z}}_2 \cdot \dot{\widehat{\mathbf{y}}}_2 \\ \widehat{\mathbf{x}}_2 \cdot \dot{\widehat{\mathbf{z}}}_2 \\ \widehat{\mathbf{y}}_2 \cdot \dot{\widehat{\mathbf{x}}}_2 \end{pmatrix}.$	



(a) (b)

Figure 6: Robot model from [5]: (a) perspective diagram; (b) kinematic scheme.

4.3 Closed-chain robot

Article [5] presents the 6 DOF industrial robot model in Figure 6 as a vehicle to teach MBS, and gives NCs kinematic equations of it. We give our, significantly shorter, version of the kinematic equations, explaining the notation but without derivations, which follow the scheme in §4.2. Simplifying assumptions are made to illustrate the "assign, don't equate" policy: ABCD (called $\mathbf{r}_2\mathbf{r}_3\mathbf{r}_4\mathbf{r}_5$ in Figure 6 (b)) is an exact parallelogram; AB is horizontal with midpoint O on the WF vertical axis of rotation OZ. Table 4 shows how DAE size increases without these assumptions.

The parameters defining the geometry are $L_{\mathsf{AB}}, L_{\mathsf{AC}}, \dots$ which are the constant lengths $|\mathsf{AB}|$, $|\mathsf{AC}|$, etc.; $\mathsf{E}_x, \mathsf{E}_z$, are E's coordinates within ⑤. As before, $\underline{\mathsf{X}}, \underline{\mathbf{u}}$ denote a fixed WF position or vector, and $\mathsf{X}, \mathbf{u}$ a moving one, and so on. Vectors $\underline{\mathbf{u}}_1$ to $\mathbf{u}_5$ are as in Figure 6(b).

In general, we want as many joints as possible to be defined by point and vector sharing. This is useful here, because subsystem ABCD becomes overconstrained if the spine directions of its four revolute joints are treated as independent. With exact alignment it can move but with small random deviations, mathematically it cannot—the issue of "paradoxical mobility", see Talaba [21] for a good discussion.

In practical manufacture this is overcome by engineering tolerances and the slight elasticity of real rigid bodies. Mathematically however, a model based on independent joint directions is statically indeterminate when these directions coincide, so the Lagrange multipliers to the constraint reactions at the joints are not uniquely determined. Numerically one might seek e.g. minimum-norm solutions using matrix singular value analysis, but this is needlessly complicated. We cut the Gordian knot, as does [5], by making the joint directions a shared vector, $\mathbf{u}_3$ in Figure 6 (b). In the resulting model the overconstrainedness simply goes away.

The frame definitions, which make each R matrix (5) diagonal, are

Part	*BPVs*	*Part*	*BPVs*
②	$\underline{\mathsf{O}}\underline{\mathbf{u}}_1\mathbf{u}_3$ with useful points A, B	③	$\mathsf{AC}\mathbf{u}_3$
④	frame not used	⑤	$\mathsf{CD}\mathbf{u}_3$ with useful point E
⑥⑦	$\mathsf{EF}\mathbf{u}_4$	⑧⑨	$\mathsf{FG}\mathbf{u}_5$

Fixed in the WF are $\underline{\mathsf{O}} = (0,0,0)^T$ and $\underline{\mathbf{u}}_1 = (0,0,1)^T$. The coordinate vector is

$$\mathbf{q} = (\mathbf{u}_3, \mathsf{C}, \mathbf{u}_4, \mathsf{F}, \mathbf{u}_5, \mathsf{G}) \qquad \text{making 18 scalars.}$$

In Table 3, "Bring in" lists the parts in the order natural to the topology, and the elements of $\mathbf{q}$ in the order they are used. "DOF" is a running count of degrees of freedom, e.g. rotating base ② on its own has 3−2=1 DOF. The parallelogram assumption implies ④, ⑤ only generate assignments without changing the DOF.

Table 3: Assignments and equations for kinematics of closed chain robot.

Bring in	Assign	Equate	DOF
	$\mathbf{u}_2 := \underline{\mathbf{u}}_1 \times \mathbf{u}_3$	$0 = \mathbf{u}_3^2 - 1$	
②; $\mathbf{u}_3$	$\mathsf{A} := \frac{1}{2} L_{\mathsf{AB}} \mathbf{u}_2$		$+3 \quad -2$
	$\mathsf{B} := -\mathsf{A}$	$0 = \mathbf{u}_3 \cdot \underline{\mathbf{u}}_1$	
③; C		$0 = (\mathsf{C}-\mathsf{A})^2 - L_{\mathsf{AC}}^2$	$+3 \quad -2$
		$0 = (\mathsf{C}-\mathsf{A}) \cdot \mathbf{u}_3$	
④	$\mathsf{D} := \mathsf{C} + \mathsf{B} - \mathsf{A}$		
⑤	$\mathsf{E} := \mathsf{C} + \mathsf{E}_x \mathbf{u}_2 + \mathsf{E}_z \underline{\mathbf{u}}_1$		
		$0 = (\mathsf{F}-\mathsf{E})^2 - L_{\mathsf{EF}}^2$	
⑥⑦; $\mathbf{u}_4, \mathsf{F}$		$0 = (\mathsf{F}-\mathsf{E}) \cdot \mathbf{u}_4$	$+6 \quad -3$
		$0 = \mathbf{u}_4^2 - 1$	
		$0 = (\mathsf{F}-\mathsf{E}) \cdot \mathbf{u}_3$	-1
		$0 = (\mathsf{G}-\mathsf{F})^2 - L_{\mathsf{FG}}^2$	
⑧⑨; $\mathbf{u}_5, \mathsf{G}$		$0 = (\mathsf{G}-\mathsf{F}) \cdot \mathbf{u}_5$	$+6 \quad -3$
		$0 = \mathbf{u}_5^2 - 1$	
		$0 = (\mathsf{G}-\mathsf{F}) \cdot \mathbf{u}_4$	-1
		Net DOF:	$6 = 18 - 12$

5 Conclusions

Summary. What we have presented has two main aspects. First, a "3 points/vectors per body" (3PVs) natural coordinates way of tracking rigid body motion that basically treats all bodies in a unified way, without the different cases in the usual kind of NCs. Second, a software architecture for MBS simulation where much of what is traditionally done explicitly, is done inside the DAE solver DAETS. This makes the mass matrix, though present, appear as a byproduct in the background—readers used to finite elements or structural mechanics might find this unusual. The

Table 4: Number of equations de Jalón–Bayo and Nedialkov–Pryce NCs (JBNCs and NPNCs) use to define kinematics of two models in the text. For RSCR: JBNCs part ② is our ①, etc. For robot: [3] notes part ④ has a redundant equation; NPNCs (a), (b) are without and with assuming ABCD is a parallelogram. DAE sizes for JBNCs in parentheses are as given to a DAE code after reduction to first-order.

1 DOF RSCR from [4, p. 73]				6 DOF closed-chain robot from [3]			
	JBNCs		NPNCs		JBNCs	NPNCs(a)	NPNCs(b)
part	# eqns	part	# eqns	part	# eqns	# eqns	# eqns
②	2	①	2	②	8	2	2
③	3	②	2	③	2	2	2
④	5	③	2	④	1 of 2	2	0
cyl. joint.	2 of 3		0	⑤	5	2	0
				⑥	2	2	2
				⑦	2	2	2
				⑧	2	2	2
				⑨	2	2	2
Total eqns	12		6		24	16	12
Length of **q**	13		7		30	22	18
DAE size	25 (38)		13		54 (84)	38	30

notorious initial position problem is also solved in the background: not because we have a magic bullet, but because the Ipopt large scale nonlinear optimiser is built into Daets.

The two aspects are related. Our Lagrangian facility works equally well for a Lagrangian based on any coordinate system: 3PVs NCs, the standard 4PVs NCs, relative coordinates, etc. The 3PVs method has the advantage of conciseness—it usually leads to a shorter coordinate vector **q** and fewer rigidity constraints—but the seeming disadvantage of a non-constant **M** matrix—inefficient on large problems when a standard DAE solver is used. In fact we think non-constant **M** shouldn't matter much because of Daets's algorithm structure: it re-uses many times each evaluation of the Jacobian **J**, of which **M** is a part.

At present, we solve small to medium systems, for which easy formulation of the model is more important than efficiency. Our many experiments have shown very efficient integrations, see [13]. We are at an early stage of overall development and objective benchmarking against other systems is not easy; we have not yet tried it.

In examples taken from de Jalón *et al.*, we do indeed get more concise models, mainly by specifying a moving 3D frame by at most 9 scalars vs 12, and by replacing equations with assignments where possible. Table 4 compares equation-counts for the RSCR model, and for the closed-chain robot of [3, 5] outlined in §4.3.

User-friendliness and potential in teaching. How easily can a new user enter their own problem into the system? A common paradigm when programming is involved, both for research and when teaching students, is to take a working example program and edit it to solve a different but related problem. We think our examples show this is straightforward for small models in the context of the

Lagrangian facility. One forms the mathematical Lagrangian and constraints by hand, and transcribes to `C++` code in a natural way as shown by examples in §4.

By contrast, use of the MechSpec—see the workflow in Figure 7—is for quick turnaround and a higher level view. The user specifies the mechanism in a YAML file, and a script automates the following steps. The YAML is read into a data structure in the `C++` main program. The part and joint specifications are interpreted and converted to a Lagrangian and constraints, passed to the Lagrangian facility which converts it to an `FCN` function suitable for DAETS. The data structure also holds initial values etc. to control the integration, from which DAETS (if all is well) finds an initial consistent point and solves to the requested end time, writing to the output data file at each time step. This and the MechSpec file are read by a MATLAB program that produces an animation of the motion.

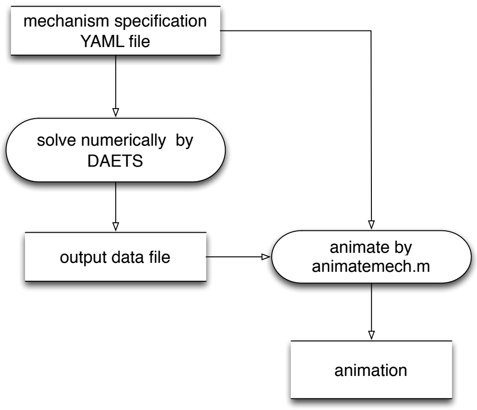


Figure 7: MBS simulation workflow.

For systems such as the examples in this article, over a modest time range, running on a laptop, one typically goes from submitting the MechSpec file to viewing the animation in a few seconds. We solved the Andrews Squeezing Mechanism up to $t = 0.3$, ten times the reference solution range, in 2.5 CPU seconds—tricky because the system, driven by a constant torque, rotates faster and faster requiring gradually reducing time steps.

De Jalón in his survey [3] of Natural Coordinates argues cogently for their use in teaching: “Teaching 3-D multibody system analysis is probably the most important application of natural coordinates, particularly when there are severe time limitations”, and [5] gives experience of putting this into practice.

We agree, and think that whether one uses a Lagrangian or a Newtonian approach to setting up the equations of motion, the extra brevity that comes from using our methods makes Natural Coordinates even more attractive for this purpose at an undergraduate or a graduate level.

Future plans. The Lagrangian facility works in both 2D and 3D. We have a well-tested 2D MechSpec and are part way through converting this to handle 3D.

Important features so far missing, and in hand, are: a general applied force $\mathbf{F}(t, \mathbf{q}, \dot{\mathbf{q}})$ applied at a point on a body, or a torque similarly; an actuator similarly, giving a prescribed linear or angular position; a Rayleigh dissipative term, often a convenient way to model damping forces within the Lagrangian. All have been tested by hand-coded 3D examples using the Lagrangian facility.

References

[1] Brenan, Kathy E., Campbell, Stephen L., and Petzold, Linda R. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* SIAM, Philadelphia, second edition, 1996. DOI: `0.1137/1.9781611971224`.

[2] Davis, Timothy A. and Palamadai Natarajan, Ekanathan. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37(3):36:1–36:17, September 2010. DOI: `10.1145/1824801.1824814`.

[3] de Jalón, Javier García. Twenty-five years of natural coordinates. *Multibody System Dynamics*, 18(1):15–33, 2007. DOI: `doi.org/crvphq`.

[4] de Jalón, Javier Garcia and Bayo, Eduardo. *Kinematic and Dynamic Simulation of Multibody Systems: The Real Time Challenge.* Springer-Verlag, Berlin, Heidelberg, 1994.

[5] de Jalón, Javier García, Shimizu, Nobuyuki, and Gómez, David. Natural coordinates for teaching multibody systems with Matlab. In *Proc. IDETC/-CIE 2007, September 4–7, 2007, Las Vegas, Nevada, USA*, volume 5, pages 1539–1548. ASME, 2007. DOI: `10.1115/DETC2007-35358`.

[6] Greiner, Walther. *Classical Mechanics: Systems of Particles and Hamiltonian Dynamics.* Number v. 1 in Classical theoretical physics. Springer, 2003. DOI: `10.1007/978-3-642-03434-3`.

[7] Hairer, E. and Wanner, G. *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems.* Springer Verlag, Berlin, second edition, 1991. DOI: `10.1007/978-3-642-05221-7`.

[8] Kraus, C., Winckler, M., and Bock, H.G. Modeling mechanical DAE using natural coordinates. *Mathematical and Computer Modelling of Dynamical Systems*, 7(2):145–158, 2001. DOI: `10.1076/mcmd.7.2.145.3645`.

[9] Li, Xiao. Incremental computation of Taylor series and system Jacobian in DAE solving using automatic differentiation. Master's thesis, Computational Science and Engineering, McMaster University, Hamilton, Ontario, Canada, 2017. DOI: `hdl.handle.net/11375/22521`.

[10] Mattsson, Sven Erik and Söderlind, Gustaf. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM J. Sci. Comput.*, 14(3):677–692, 1993. DOI: `10.1137/0914043`.

[11] Nedialkov, Nedialko S and Pryce, John D. Solving differential-algebraic equations by Taylor series (III): the DAETS code. *JNAIAM J. Numer. Anal. Indust. Appl. Math*, 3:61–80, 2008. DOI: `jnaiamcont.org/new/uploads/files/b4e0bdaa7990bbd1896fa3fafdd09f71.pdf`.

[12] Nedialkov, Nedialko S. and Pryce, John D. DAETS user guide. Technical Report CAS 08-08-NN, Department of Computing and Software, McMaster University, Hamilton, ON, Canada, June 2013. 68 pages, DAETS is available at `http://www.cas.mcmaster.ca/~nedialk/daets`.

[13] Nedialkov, Nedialko S. and Pryce, John D. Multi-body Lagrangian simulations, 2017. YouTube channel, `https://www.youtube.com/channel/UCCuLchOxOWOyoNE9KOCYlVQ`.

[14] Nedialkov, Nedialko S. and Pryce, John D. YAML specification of 2D mechanisms for the DAETS Lagrangian facility. Technical report, Department of Computing and Software, McMaster University, 2018. In preparation.

[15] Nikravesh, Parviz E. An overview of several formulations for multibody dynamics. In Talabă, D. and Roche, T., editors, *Product Engineering*, pages 181–207. Springer, Dordrecht, 2004. DOI: `doi.org/bj6272`.

[16] OpenSim. OpenSim implementation of MBS Benchmark, 2008 (accessed July 2017). `rehabenggroup.github.io/MBSbenchmarksInOpenSim/index.html`.

[17] Pantelides, Costas C. The consistent initialization of differential-algebraic systems. *SIAM J. Sci. Stat. Comput.*, 9:213–231, 1988. DOI: `10.1137/0909014`.

[18] Pryce, John D. A simple structural analysis method for DAEs. *BIT Numerical Mathematics*, 41(2):364–394, 2001. DOI: `10.1023/A:1021998624799`.

[19] Pryce, John D., Nedialkov, Nedialko S., Tan, Guangning, and Li, Xiao. How AD can help solve differential-algebraic equations. *Optimization Methods and Software*, 33(4-6):729–749, 2018. DOI: `10.1080/10556788.2018.1428605`.

[20] Susskind, L. and Hrabovsky, G. *Classical Mechanics: The Theoretical Minimum*. The theoretical minimum. Penguin Books, 2014. DOI: `10.1119/1.4816681`.

[21] Talabă, Doru. Mechanical models and the mobility of robots and mechanisms. *Robotica*, 33(1):181–193, 2015. DOI: `10.1017/S0263574714000149`.

[22] von Schwerin, Reinhold. *Multibody system simulation: numerical methods, algorithms, and software*, volume 7. Springer Science & Business Media, 2012. DOI: `10.1007/978-3-642-58515-9`, Softcover reprint of the original 1st ed. 1999.

[23] Wächter, Andreas and Biegler, Lorenz T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006. DOI: `10.1007/s10107-004-0559-y`.