DATA, LAGRANGIAN, ACTION! SIMULATING MECHANISMS DIRECT FROM A TEXT FILE*

JOHN D. PRYCE† AND NEDIALKO S. NEDIALKOV‡

Abstract. We describe a representation of linkage mechanisms in Lagrangian form using cartesian coordinates, and an implementation thereof. As seen by a user, it is a tool that reads a text data file specifying a mechanism along with initial values and integration range, solves the resulting initial-value problem, and produces an animation of the result. The file is in the YAML language.

The underlying theory, for a rigid body $\mathcal{R}$ moving in n dimensions, uses QR factorisation to give the position and velocity of an arbitrary point on $\mathcal{R}$ as a function (linear in 2D, bilinear in 3D, etc.) of n reference points (RPs) fixed on $\mathcal{R}$. For 2D and 3D we derive formulae for $\mathcal{R}$'s kinetic energy, in terms of velocities of the RPs and of constants of $\mathcal{R}$—centroid, mass, moments of inertia.

Summing over the parts of a mechanism, including potential energy and applied force terms, gives the total Lagrangian $\mathcal{L}$ as a function of the RPs in cartesian coordinates. Rigidity, and joints, define algebraic constraints. The *mechanism facility* (currently 2D only) automates the conversion of the data file to $\mathcal{L}$. The *Lagrangian facility* (2D or 3D) converts $\mathcal{L}$ to the equations of motion, by algorithmic differentiation at run time without computer algebra.

The result is a differential-algebraic equation system (DAE) of index 3, because of the constraints. High-index DAEs have been seen as hard to solve numerically, so this has been considered an impractical route to solution, and much work has gone into representing $\mathcal{L}$ in constraint-free coordinates, typically angles, leading to an ODE system. Efficient high-index DAE initial value solvers change the balance of advantage. We use DAETS, which solves this kind of DAE as easily as an ODE.

The paper presents the theory, illustrates the steps file → Lagrangian → DAE → solution by examples in 2D and 3D, and reports numerical experiments and animations viewable online.

Key words. Mechanisms, Lagrangian mechanics, simulation, differential-algebraic equations, algorithmic differentiation, YAML

AMS subject classifications. 65L05, 65L80, 70E55, 70H03

1. Introduction.

1.1. Context and aims.

We are concerned with linkage mechanisms, built from parts such as rigid bodies and springs, usually connected by various kinds of joint. For simulation, a Lagrangian approach [3, 13] to forming their equations of motion is popular because of its economy and flexibility.

The economy is because Lagrangians, in contrast to direct use of Newton's three laws, can omit mention of certain forces. By d'Alembert's principle of virtual work, this amounts to declaring such forces do no work: individually, e.g. reaction force under smooth sliding; or as a group, e.g. string tensions either side of a smooth pulley. This usually means relevant system components are frictionless.

Flexibility is due to the freedom to choose *generalised coordinates* $\mathbf{q} = (q_1, \ldots, q_{n_q})$ to specify system configuration. Consider the basic case where the only forces acting are conservative so that they can be regarded as due to a potential energy field. Among all possible motions of the system, the actual one is a stationary point of the

*June 11, 2018.

Funding: N. Nedialkov was funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

†School of Mathematics, Cardiff University, Cardiff CF24 4AG, United Kingdom (PryceJD1@cardiff.ac.uk).

‡Department of Computing and Software, McMaster University, Hamilton, Canada, L8S 4K1 (nedialk@mcmaster.ca)

action integral $\int \mathcal{L}\mathrm{d}t$ of the Lagrangian function

$$\mathcal{L} = T - V, \tag{1.1}$$

where T is the system's kinetic energy and V its potential energy.

The function $\mathcal{L}$ is an inherent system property, independent of the chosen coordinates $\mathbf{q}$. For any $\mathbf{q}$, applying the Euler–Lagrange conditions for stationarity leads to equations of motion

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial \mathcal{L}}{\partial \dot{q}_j} - \frac{\partial \mathcal{L}}{\partial q_j} + \sum_{i=1}^{n_c} \lambda_i \frac{\partial C_i}{\partial q_j} = 0, \quad j = 1, \ldots, n_q, \tag{1.2}$$

$$C_i = 0, \quad i = 1, \ldots, n_c. \tag{1.3}$$

Here T and V are assumed to have the form $T(t, \mathbf{q}, \dot{\mathbf{q}})$ and $V(t, \mathbf{q})$. The n_c equations $C_i(t, \mathbf{q}) = 0$ are any constraints on the motion due to rigidity etc., with associated Lagrange multipliers λ_i. We assume them holonomic, i.e. C_i does not depend on $\dot{\mathbf{q}}$.

For a DAE (1.2, 1.3), the number of positional and of velocity degrees of freedom (DOF) are the same, and equal

$$\textsc{dof} = n_q - n_c. \tag{1.4}$$

(We write DOF for the concept and DOF for the number.) So the system state is locally fixed at any time by DOF values q_j and DOF values $\dot{q}_j$.

If constraints are absent ($n_c = 0$) then (1.2, 1.3) 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. The DAE has index 3, in the classical differential index sense [2], when solved as an initial-value problem (IVP).

High-index DAEs have been seen as hard to solve numerically, so much effort has gone into making $n_c = 0$, i.e. finding representation methods that lead to constraint-free coordinates, typically angles. An efficient *high-index DAE solver* changes the balance of advantage. We use the `C++` code Daets [6, 7], which solves a constrained Lagrangian DAE as easily as an ODE.

On top of Daets we have built the *Lagrangian facility*. Operationally, its key innovation is that the user just provides code for Lagrangian (1.1) and constraints (1.3). From these it constructs the equations of motion (1.2) and hence the complete DAE (1.2, 1.3) for Daets to solve—entirely by *algorithmic differentiation* (AD) at run time [11]. There is no symbolic manipulation, e.g. by a computer algebra system.

On top of this again is built the *mechanism facility*. At run time it reads a "MechSpec" file describing a mechanism. It uses a $\mathbf{q}$ made (except see §4.3) of world-frame cartesian coordinates of selected points on the parts. For this $\mathbf{q}$ it constructs the corresponding equations (1.1) and (1.3), which are passed to the Lagrangian facility.

The Lagrangian facility works in 2D and 3D. An initial 2D version of the mechanism facility has been completed: see §4.1 for a list of features.

The text is structured as follows. The rest of Section 1 outlines the methodology and gives an example to support our claim about the balance of advantage. Sections 2 and 3 give the theory of tracking rigid motion in cartesian coordinates. Section 4 explains the mechanism facility data format—the MechSpec—by 2D examples with numerical results. Section 5 outlines how we animate in MATLAB™. Section 6 summarises what we have done on the 3D case. Section 7 gives conclusions. An appendix shows a sample complete data file.

1.2. Methodology outline.

This summarises the process developed in Section 3 whereby (1.1, 1.3) are derived from mechanism data.

Notation: The style $\mathbf{q}$ is used for a generalised-coordinates vector; $\boldsymbol{x}$ for a position vector in a world or local frame; x for a displacement, a difference of position vectors.

1.2.1. *Basic idea.* In n-dimensional Euclidean space, let $\mathcal{R}$ be a rigid body moving continuously with respect to a fixed newtonian frame of reference, the *world-frame* (WF for short). Then its position—that is, the WF position of any point $\boldsymbol{x}$ on it—is determined by that of n fixed *reference points* (RPs) on it in "general position".

1.2.2. *Topology and local frames.* For each rigid part of a mechanism, fix a Euclidean coordinate system, the *local frame*. Our convention is that the frame is defined by the ordered list of the part's RPs as described in §3.1.1.

The MechSpec tabulates the cartesian coordinates of the RPs, and of each *useful point* (UP), typically centroid and points where a force is applied. It also records definitions of each joint between parts.

1.2.3. *Coordinates.* A mechanism's moving RPs (those not fixed in the WF) are coordinate points $\boldsymbol{q}_j(t)=(x_j(t), y_j(t))$ or $(x_j(t), y_j(t), z_j(t))$, concatenated to form the generalised coordinate vector $\mathbf{q}(t)$, such that the total kinetic and potential energies and the effect of applied forces, hence the Lagrangian, can be expressed in terms of $\mathbf{q}$ and $\dot{\mathbf{q}}$. (But see §4.3.)

1.2.4. *Position relations.* For each part $\mathcal{R}$ of a mechanism we fix RPs as above, and precompute constant data that gives the position of its UPs in terms of the RP positions, in $\mathcal{R}$'s frame. We do this by QR-factorising a matrix whose columns are $\mathcal{R}$'s RPs followed by its UPs. This gives the WF position of each UP, at any time t, in terms of the RP positions. The dependence of a UP on the RPs is *linear* in 2D, *bilinear* in 3D, and so on in higher dimensions.

1.2.5. *Velocity relations.* Rigid motion is by definition a combination of a t-dependent translation by a vector $\boldsymbol{p}_t$ and a t-dependent rotation Q_t. Differentiating gives the velocity of each UP of each part, including the centroid, which will be a UP, in terms of RP velocities; also the derivative $\dot{Q}_t$. The skew matrix $Q_t^T\dot{Q}_t$ gives the part's angular velocity, representable by a scalar in 2D, a 3-vector in 3D.

1.2.6. *Lagrangian.* Hence, with mass and moment of inertia data from the MechSpec, we find each part's linear and rotational kinetic energy contribution to the Lagrangian. Other contributions, such as potential energy and external forces, are also expressed in terms of $\mathbf{q}$.

1.2.7. *Constraints.* Suitable equations for rigid parts are added to give a constrained Lagrangian system defining the motion. These say two RPs on a body are a constant distance apart, a UP is correctly placed relative to the RPs, for a revolute joint the joint-spines in each body line up, etc.

1.2.8. *Mechanism and Lagrangian facilities.* These allow the above to be parameterised, so creation of the constrained Lagrangian system is not hardcoded in a program, but driven by a MechSpec file and converted on the fly to the Euler–Lagrange DAE in the form DAETS requires. How they are implemented is not described here.

Finally DAETS solves this DAE, using initial values (IVs) and a time range given in the MechSpec. At each time step the $\mathbf{q}$ vector is written to an output file, which can be passed to the animation tool and optionally be made into a movie.

1.3. Example: cartesian versus angle coordinates. Consider a multi-pendulum, a chain of N particles (bobs) all of mass m, joined by N massless rigid rods all of length ℓ, hung from a fixed point and moving under the acceleration of gravity g. The joints between rods are free pivots, i.e. their only constraint is that the start of one rod coincides with the end of the previous (or with the top pivot), so in the 3D case each joint contributes 2 DOF, totalling $\text{DOF} = 2N$.

The simplest case, namely 2D and $N = 1$, is the simple pendulum. If coordinates are the x, y position of the bob, with $+x$ horizontal to the right and $+y$ upward (following our convention later), the Lagrangian method leads after simplifying to the index-3 DAE on the left of (1.5). If the coordinate is the angle θ made by the rod with the downward vertical, it gives the ODE on the right:

$$\left.\begin{array}{lcl} \ddot{x} + x\lambda & = & 0 \\ \ddot{y} + y\lambda + g & = & 0 \\ x^2 + y^2 - \ell^2 & = & 0 \end{array}\right\}, \qquad \ddot{\theta} + \frac{g}{\ell}\sin\theta = 0. \tag{1.5}$$

Here, the angle form is simpler, but for $N > 1$ and motion in 3D, the balance of advantage shifts.

Namely for $N > 1$ in cartesian coordinates, take the absolute positions $\mathbf{r}_i = (x_i, y_i, z_i)$ of the bobs with z upward for $i = 1, \ldots, N$, giving generalised coordinate vector $\mathbf{q} = (\mathbf{r}_1, \ldots, \mathbf{r}_N) = (x_1, y_1, z_1, \ldots, x_N, y_N, z_N)$. This system's Lagrangian in (1.6) is an obvious extension of the one that led to the left of (1.5). It sums kinetic and potential energy contributions from the bobs, and the N constraints say the rods have length ℓ:

$$\left.\begin{array}{rl} \mathcal{L} = \frac{1}{2}m\sum_{i=1}^{N} |\dot{\mathbf{r}}_i|^2 - mg\sum_{i=1}^{N} z_i, & \\ 0 = C_i = |\mathbf{r}_i - \mathbf{r}_{i-1}|^2 - \ell^2, & i = 1, \ldots, N, \end{array}\right\}, \tag{1.6}$$

with the convention that $\mathbf{r}_0$ is constant equal to $\mathbf{0}$. The resulting DAE has $3N$ coordinate variables and N Lagrange multipliers, so is second-order of size $4N$. DAETS handles it directly—it does not need to reduce to first-order form.

For an angle formulation, about the simplest method is to use spherical polar coordinates (θ_i, ϕ_i) for rod i, where θ_i is the rod's angle with the downward vertical and ϕ_i is angle of rotation from the xz plane, for $i = 1, \ldots, N$. The resulting generalised coordinate vector $\mathbf{q} = (\theta_1, \phi_1, \ldots, \theta_N, \phi_N)$ gets rid of the constraints, and produces an ODE system, of size $4N$ when reduced to first order as most ODE codes require.

However this $\mathbf{q}$ gives the position of each bob *relative* to the preceding one. One must sum these to reconstruct absolute positions and velocities and thus compute the energies T and V. E.g.,

$$T = \frac{1}{2}m\ell^2 \sum_{k=1}^{n} \left| \sum_{i=1}^{k} \begin{pmatrix} \cos\theta_i\, \dot{\theta}_i \cos\phi_i - \sin\theta_i \sin\phi_i\, \dot{\phi}_i \\ \cos\theta_i\, \dot{\theta}_i \sin\phi_i + \sin\theta_i \cos\phi_i\, \dot{\phi}_i \\ -\sin\phi_i\, \dot{\phi}_i \end{pmatrix} \right|^2$$

where the $|\cdot|^2$ is the squared length of a 3-vector. It seems any other way to remove the constraints will use angles in some form, and be similarly cumbersome.

This example suggests a constrained cartesian formulation is often simpler than a constraint-free one, as well as sparser (fewer variables per equation). Thus the DAE may be more economical overall to solve than is the ODE. Indeed DAETS has quickly solved instances of (1.6) with 40 pendula, even when the motion is quite chaotic.

2. Some linear algebra.

2.1. General cross product. Computing angular velocities involves the *cross product*—essentially the "wedge" product of $n-1$ vectors in n-space, in exterior algebra [1]. We define it here as the way a determinant depends on one of its columns.

For an $n\times(n-1)$ matrix W with columns $w_1,\dots,w_{n-1}$, the map $e\mapsto \det\big[W,e\big]$, the determinant of the result of appending $e\in\mathbb{R}^n$ to W, is a linear functional.

DEFINITION 2.1. *The* cross product *of W, or of the w_j, is the vector that implements this functional, i.e. the unique $c\in\mathbb{R}^n$ such that*

$$c^T e = \det\big[W,e\big], \qquad e\in\mathbb{R}^n. \tag{2.1}$$

We write it in two equivalent notations (the second only applies when $n\ge 3$):

$$c = \bigtimes W = w_1\times w_2\times\cdots\times w_{n-1}.$$

Clearly c is orthogonal to each w_j, since if e equals any w_j, determinant (2.1) has two equal columns and vanishes. One may view c as the result of expanding the determinant treating the components of e as symbols, $\det\big[W,e\big]=c_1e_1+\cdots+c_ne_n$, and then letting them be the standard basis vectors, $[1,0,\dots,0]^T$, $[0,1,\dots,0]^T$, etc.

Two cases concern us.

Case $n=2$: Let the column of W be $u=\begin{bmatrix}u_1\\u_2\end{bmatrix}$, then

$$\bigtimes W = \det\big[W,e\big] = \det\begin{bmatrix}u_1 & e_1\\ u_2 & e_2\end{bmatrix} = -u_2e_1+u_1e_2 = \begin{bmatrix}-u_2\\ u_1\end{bmatrix},$$

which is the vector u rotated through a positive right angle.

Case $n=3$: Write $W=\big[u\quad v\big]=\begin{bmatrix}u_1 & v_1\\ u_2 & v_2\\ u_3 & v_3\end{bmatrix}$, then

$$\bigtimes W = \det\begin{bmatrix}u_1 & v_1 & e_1\\ u_2 & v_2 & e_2\\ u_3 & v_3 & e_3\end{bmatrix} = (u_2v_3-u_3v_2)e_1+(u_3v_1-u_1v_3)e_2+(u_1v_2-u_2v_1)e_3$$
$$= \begin{bmatrix}u_2v_3-u_3v_2\\ u_3v_1-u_1v_3\\ u_1v_2-u_2v_1\end{bmatrix}, \qquad \text{the usual 3D cross product } u\times v.$$

(This determinant is a common mnemonic for the 3D cross product formula.)

THEOREM 2.2. *For any nonsingular $n\times n$ matrix Q*

$$\bigtimes(QW) = \det(Q).(Q^T)^{-1}(\bigtimes W). \tag{2.2}$$

In particular if Q is a rotation,

$$\bigtimes(QW) = Q(\bigtimes W). \tag{2.3}$$

Proof. Let $c=\bigtimes W$ and $d=\bigtimes(QW)$. For any e, from the definition of $\bigtimes$, matrix multiplication being associative, and the fact $\det(AB)=\det(A)\det(B)$ we have

$$(d^TQ)e = d^T(Qe) = \det\big[QW,Qe\big] = \det(Q\big[W,e\big]) = \det(Q)\det(\big[W,e\big]) = \det(Q)c^Te.$$

But e is arbitrary, so $d^TQ=\det(Q)c^T$, that is $Q^Td=\det(Q)c$, or $d=\det(Q)(Q^T)^{-1}c$, proving (2.2). A rotation has $\det(Q)=1$ and $Q^{-1}=Q^T$, whence (2.3). ∎

COROLLARY 2.3. *For a rotation, the cross product of columns $1{:}n-1$ is column n.*

Proof. This is trivial for the identity; then (2.3) proves the general case. ∎

2.2. Complexity aspects. The complexity of computing $c = \bigtimes W$ for a general n is of interest, even though below we only consider $n \le 3$. Suppose W is $n \times (n-1)$. An $O(n^3)$ method is as follows. Factorise $W = QR$ where Q is a rotation and R is upper triangular. Then by Theorem 2.2, c is the last column of Q times the product of R's diagonal elements.

If the w_j are already known to be the first $n-1$ columns of a rotation, one can find $\pm c$ in $O(n^2)$ time by orthonormalising a "random" vector with respect to the w_j. It seems less obvious how to choose the correct sign for c and we suspect no method better than $O(n^3)$ exists.

3. Motion computed from reference points in n dimensions. We derive the formulae for general dimension n, though only $n = 2$ or 3 are of direct relevance. E^n denotes n-dimensional Euclidean space with some point O chosen as origin; we identify points with their position-vectors relative to O, making E^n a real linear space, with an inner product defining the geometry. E^n can be identified with $\mathbb{R}^n$ with its usual inner product, when a *frame*—an origin plus an orthonormal basis of coordinates—is chosen. *Fixed space* has a distinguished newtonian inertial frame, the *world-frame* WF, in which other parts and their frames move.

As in §1.2 we use the style $\boldsymbol{a}$ for position-vectors in a frame, and a for displacements, differences of position vectors.

3.1. Positions and velocities of a part. To follow the mission in §1.2, we consider one rigid part $\mathcal{R}$ of a mechanism at a time, and seek an algorithm that does:

(3.1)
- (a) *Express the position of $\mathcal{R}$'s centroid and any other UPs at current time t, as functions of the current RP positions and precomputed constants.*
- (b) *By differentiating, express UP velocities and $\mathcal{R}$'s angular velocity in terms of RP velocities and precomputed constants; hence find $\mathcal{R}$'s kinetic energy contribution to the Lagrangian.*

3.1.1. Positions. A rigid body $\mathcal{R}$ is given, moving in the WF. Rigid means its position at time t is

$$\mathcal{R}_t = \boldsymbol{p}_t + Q_t\mathcal{R} \tag{3.2}$$

for some point $\boldsymbol{p}_t$ and rotation Q_t, both being functions of t. On the right, $\mathcal{R}$ denotes a fixed reference version of the body in a chosen *local frame*. A *rotation* is a member of the group $\mathrm{SO}(n)$ of orthogonal transformations on E^n with determinant $+1$. Each point $\boldsymbol{x}$ in $\mathcal{R}$ moves according to (3.2): its WF position at time t is $\boldsymbol{x}_t = \boldsymbol{p}_t + Q_t\,\boldsymbol{x}$.

Let there be n *reference points* (RPs) fixed in $\mathcal{R}$

$$\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1},$$

and p further *useful points* (UPs) fixed in $\mathcal{R}$

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p.$$

The RPs shall be in general position, i.e. there is a unique hyperplane (translate of an $(n-1)$-dimensional linear subspace) on which they lie. A UP can be anywhere. The union of the RPs and UPs will be called the reference-or-useful points (RUPs).

With $\boldsymbol{a}_0$ as *base* reference point, define *displacements* of the other RUPs from it:

$$a_j = \boldsymbol{a}_j - \boldsymbol{a}_0,\ (j = 1, \ldots, n-1), \quad x_j = \boldsymbol{x}_j - \boldsymbol{a}_0,\ (j = 1, \ldots, p). \tag{3.3}$$

The next fact is well known.

LEMMA 3.1. *The following are equivalent:*
(i) $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1}$ *are in general position.*
(ii) $a_1, \ldots, a_{n-1}$ *are linearly independent.*
(iii) The vectors $\begin{bmatrix}\boldsymbol{a}_0\\1\end{bmatrix}, \ldots, \begin{bmatrix}\boldsymbol{a}_{n-1}\\1\end{bmatrix}$ *are linearly independent in* $(n+1)$*-space.*

Without a suffix "$_t$", items in (3.3) denote constants in $\mathcal{R}$. With it, e.g. $\boldsymbol{a}_{1,t}$, they are varying in the WF. Write them as $\mathbb{R}^n$-vectors using an arbitrary but fixed *local frame* in $\mathcal{R}$, and form $n \times (n-1)$ and $n \times p$ matrices

$$A = [a_1, a_2, \ldots, a_{n-1}], \qquad X = [x_1, x_2, \ldots, x_p], \tag{3.4}$$

The MechSpec data format of §4 makes the rule:

(3.5) *Frame convention:* reference points define the local frame.

This means, in traditional xyz notation, that $\boldsymbol{a}_0$ is the origin; line $\boldsymbol{a}_0\boldsymbol{a}_1$ is the x-axis with $\boldsymbol{a}_1$ on the positive side, $\boldsymbol{a}_0\boldsymbol{a}_1\boldsymbol{a}_2$ define the xy-plane with $\boldsymbol{a}_2$ in the upper half-plane, and so on in any higher dimensions. Equivalently, in the coordinates thus defined, matrix A in (3.4) is upper triangular with $a_{ii} > 0$. The direction of the final axis (y if in 2D, z if in 3D, ...) is chosen to give the frame positive orientation in the whole space, i.e. $a_{nn} > 0$. Below, this makes $\overline{A} = A$ and $\overline{Q}$ = identity, but the theory in the rest of §3 does not need this.

We form a QR factorisation of $[A, X]$:

$$[A, X] = \overline{Q}\,[\overline{A}, \overline{X}]. \tag{3.6}$$

Here $\overline{Q}$ is $n \times n$ orthogonal; $\overline{A}, \overline{X}$ have the same size as A, X; and we can write

$$\overline{A} = \begin{bmatrix}\overline{R}\\0\end{bmatrix},$$

with $\overline{R} = [r_{ij}]$ square upper triangular of size $n-1$. By Lemma 3.1 A's columns are linearly independent, so $\overline{R}$ is nonsingular, so its diagonal elements r_{ii} are nonzero. We can, and do, ensure that (a) all $r_{ii} > 0$; (b) $\det \overline{Q} = +1$ so that $\overline{Q}$ is a rotation.

Let the columns of $\overline{Q}$ be $u_1, \ldots, u_n$, forming an orthonormal basis of $\mathbb{R}^n$, and let $\overline{P}$ be its first $n-1$ columns so that

$$\overline{Q} = [\overline{P}, u_n]. \tag{3.7}$$

Because the bottom row of $\overline{A}$ is zero, we have $A = \overline{P}\overline{R}$, hence

$$\overline{P} = A\,\overline{R}^{-1}. \tag{3.8}$$

Note that $\overline{Q}$, $\overline{A}$ and $\overline{X}$, hence $\overline{P}$ and $\overline{R}$, are constants, computed in $\mathcal{R}$'s frame before motion starts. We use $\overline{Q}$ in the graphics, see §5. In 2D it is not used in the numerical solution; in 3D it appears in (3.16) but could be eliminated by replacing constants $\mathbf{I}_{\boldsymbol{c}}$ with $\widehat{\mathbf{I}}_{\boldsymbol{c}} = \overline{Q}^T \mathbf{I}_{\boldsymbol{c}} \overline{Q}$, and ω with $\widehat{\omega}$, in (3.17).

Under rigid motion, the RPs and UPs move according to (3.2). Following (3.3) we have moving displacements

$$a_{j,t} = \boldsymbol{a}_{j,t} - \boldsymbol{a}_{0,t},\ (j = 1, \ldots, n-1), \quad x_{j,t} = \boldsymbol{x}_{j,t} - \boldsymbol{a}_{0,t},\ (j = 1, \ldots, p).$$

forming matrices A_t and X_t corresponding to (3.4). We wish to express the $x_{j,t}$ as functions of the $a_{j,t}$, hence X_t in terms of A_t. On forming displacements, the translation $\boldsymbol{p}_t$ in (3.2) cancels, so that $a_{j,t} = Q_t a_j$ and $x_{j,t} = Q_t x_j$. Equivalently

$$A_t = Q_t\, A, \quad X_t = Q_t\, X. \tag{3.9}$$

Using (3.9) and then (3.6), we have

$$\begin{aligned} [A_t, X_t] &= Q_t\,[A, X] = Q_t \overline{Q}\,[\overline{A}, \overline{X}] \\ &= \widehat{Q}_t\,[\overline{A}, \overline{X}], \qquad\qquad \text{where } \widehat{Q}_t = Q_t\overline{Q}, \end{aligned} \tag{3.10}$$

so $\widehat{Q}_t$, being a product of rotations, is a rotation. Let its columns be $\widehat{u}_{1,t}, \dots, \widehat{u}_{n,t}$,. Similarly to (3.7), let $\widehat{P}_t$ be its first $n-1$ columns. Arguing as in (3.7, 3.8) we have

$$\widehat{P}_t = A_t\, \overline{R}^{-1}.$$

That is, from the known A_t and the precomputed triangular matrix $\overline{R}$, we reconstruct all of $\widehat{Q}_t$ except its last column $\widehat{u}_{n,t}$. This now equals $\bigtimes \widehat{P}_t$ by Corollary 2.3. Finally compute X_t from (3.10), and obtain the $\boldsymbol{x}_{j,t}$ by adding back the base position from which the displacements $x_{j,t}$ were calculated in (3.3).

Write $A \ominus x$ for the result of subtracting vector x from each column of matrix A; similarly $A \oplus x$. Then (3.1) item *(a)*, expressing UP positions as functions of RP positions, is summarised thus:

$$\left\{\begin{array}{l} \text{A. Precomputed constant items} \\ \qquad A = \big[\boldsymbol{a}_1, \dots, \boldsymbol{a}_{n-1}\big] \ominus \boldsymbol{a}_0, \quad X = \big[\boldsymbol{x}_1, \dots, \boldsymbol{x}_p\big] \ominus \boldsymbol{a}_0, \\ \qquad \big[A, X\big] = \overline{Q}\,\left[\begin{bmatrix}\overline{R}\\ 0\end{bmatrix}, \overline{X}\right], \quad \overline{Q} \text{ a rotation, } \overline{R} \text{ upper triangular} \\ \text{B. During numerical solution} \\ \qquad A_t = \big[\boldsymbol{a}_{1,t}, \dots, \boldsymbol{a}_{n-1,t}\big] \ominus \boldsymbol{a}_{0,t}, \\ \qquad \widehat{P}_t = A_t\, \overline{R}^{-1}, \\ \qquad \widehat{Q}_t = \left[\widehat{P}_t, \quad \bigtimes \widehat{P}_t\right], \\ \qquad X_t = \widehat{Q}_t\, \overline{X}, \\ \big[\boldsymbol{x}_{1,t}, \dots, \boldsymbol{x}_{p,t}\big] = X_t \oplus \boldsymbol{a}_{0,t}. \end{array}\right. \tag{3.11}$$

3.1.2. Linear velocities. The $\boldsymbol{x}_{j,t}$ depend linearly on the $\boldsymbol{a}_{j,t}$ except for the cross product, which in n dimensions is $(n-1)$-linear. Let a general t-dependent $n \times (n-1)$ matrix W have columns $w_1, \dots, w_{n-1}$. Then $\bigtimes W = w_1 \times w_2 \times \cdots \times w_{n-1}$ differentiates as usual for a multilinear function:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(\bigtimes W) =& (\dot{w}_1 \times w_2 \times \cdots \times w_{n-1}) + (w_1 \times \dot{w}_2 \times \cdots \times w_{n-1}) \\ &+ \cdots + (w_1 \times w_2 \times \cdots \times \dot{w}_{n-1}), \end{aligned} \tag{3.12}$$

where in the jth term on the right, w_j is differentiated and the other factors are not. So the $\dot{\boldsymbol{x}}_{j,t}$ come from differentiating (3.11)B step by step, handling $\bigtimes \widehat{P}_t$ by (3.12).

3.1.3. Angular velocity and kinetic energy. The kinetic energy T of $\mathcal{R}$ is expressed in terms of: the WF velocity $\dot{\boldsymbol{c}}$ of its centroid $\boldsymbol{c}$; its angular velocity ω; and

its total mass M and its moment of inertia $\mathbf{I_c}$ about the centroid. By (3.10) we have $Q_t = \widehat{Q}_t \overline{Q}^T$ so, since $\overline{Q}$ is constant,

$$\dot{Q}_t = \dot{\widehat{Q}}_t \overline{Q}^T. \tag{3.13}$$

Differentiating $Q_t^T Q_t = I$ shows $Q_t^T \dot{Q}_t$ is a skew-symmetric matrix S_t, in 1-1 correspondence with the angular velocity data ω. In the notation of [12], write it as $[\![\omega]\!]$:

$$[\![\omega]\!] = Q_t^T \dot{Q}_t = \overline{Q}\,(\widehat{Q}_t^T \dot{\widehat{Q}}_t)\,\overline{Q}^T \qquad \text{by (3.13).} \tag{3.14}$$

Case $n = 2$: Here

$$S_t = [\![\omega]\!] = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix},$$

where ω is the scalar angular velocity. Now S_t commutes with rotations, in particular with $\overline{Q}$. Hence by (3.14) we can find ω from computed $\widehat{Q}_t$ instead of theoretical Q_t:

$$[\![\omega]\!] = \widehat{Q}_t^T \dot{\widehat{Q}}_t. \tag{3.15}$$

The kinetic energy is then given by

$$T = \tfrac{1}{2} M\,|\dot{\boldsymbol{c}}|^2 + \tfrac{1}{2} \mathbf{I_c}\,\omega^2.$$

Case $n = 3$: Here

$$S_t = [\![\boldsymbol{\omega}]\!] =_{\text{def}} \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix},$$

where $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T$ is the angular velocity vector. This matrix "does cross product" in the sense that $[\![\boldsymbol{\omega}]\!]\, x = \boldsymbol{\omega} \times x$ for any 3-vector x.

Again we can find $\boldsymbol{\omega}$ from the computed $\widehat{Q}_t$ instead of the theoretical Q_t. Namely in (3.14), $\widehat{Q}_t^T \dot{\widehat{Q}}_t$ is also skew, call it $[\![\widehat{\boldsymbol{\omega}}]\!]$. Then Theorem 2.2 shows

$$\boldsymbol{\omega} = \overline{Q}\,\widehat{\boldsymbol{\omega}}. \tag{3.16}$$

The kinetic energy is then given by

$$T = \tfrac{1}{2} M\,|\dot{\boldsymbol{c}}|^2 + \tfrac{1}{2}\,\boldsymbol{\omega}^T \mathbf{I_c}\,\boldsymbol{\omega}, \tag{3.17}$$

with $\mathbf{I_c}$ the symmetric positive semi-definite moment of inertia matrix of $\mathcal{R}$ about $\boldsymbol{c}$.

3.2. The 2D case.

In 2D, take the case where there is one UP, which later will be $\mathcal{R}$'s centroid. Rename the two RPs and UP, ($\boldsymbol{a}_0$, $\boldsymbol{a}_1$, $\boldsymbol{x}_1$), as the corners ($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$) of a triangle. Use the traditional cyclic-order notation for triangles, see Figure 1, namely (all items are constants in the $\mathcal{R}$ frame) let

$$a = \boldsymbol{c} - \boldsymbol{b}, \quad b = \boldsymbol{a} - \boldsymbol{c}, \quad c = \boldsymbol{b} - \boldsymbol{a}.$$

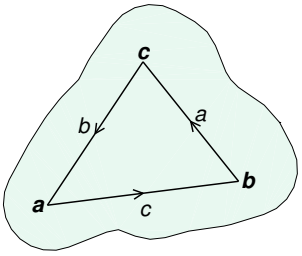


FIG. 1. *Triangle notation in 2D.*

and similarly for the moving items: $a_t = \boldsymbol{c}_t - \boldsymbol{b}_t$, $b_t = \boldsymbol{a}_t - \boldsymbol{c}_t$, $c_t = \boldsymbol{b}_t - \boldsymbol{a}_t$. We wish to express $\boldsymbol{c}_t$ as a function of $\boldsymbol{a}_t$, $\boldsymbol{b}_t$ and precomputed constants, and similarly for $\dot{\boldsymbol{c}}_t$.

Position. It is left as an exercise that the steps of (3.11) become

$$\left\{\begin{array}{l} \text{A. Precomputed constant items} \\ \qquad \overline{\xi} = \dfrac{1}{|c|^2}\begin{bmatrix} -c_1 b_1 - c_2 b_2 \\ c_2 b_1 - c_1 b_2 \end{bmatrix}. \\ \text{B. During numerical solution} \\ \qquad c_t = \boldsymbol{b}_t - \boldsymbol{a}_t, \quad \boldsymbol{c}_t = \boldsymbol{a}_t + \left[c_t, \bigtimes c_t\right]\overline{\xi}. \end{array}\right. \tag{3.18}$$

Velocity. For $n = 2$, dependence of UPs on RPs is linear so, differentiating (3.18),

$$\dot{c}_t = \dot{\boldsymbol{b}}_t - \dot{\boldsymbol{a}}_t, \quad \dot{\boldsymbol{c}}_t = \dot{\boldsymbol{a}}_t + \left[\dot{c}_t, \bigtimes \dot{c}_t\right]\overline{\xi}. \tag{3.19}$$

Angular velocity and KE. Extracting the (2,1) element from $[\![\omega]\!]$ in (3.15), with the current notation, gives

$$\omega = (-c_{2,t}\,\dot{c}_{1,t} + c_{1,t}\,\dot{c}_{2,t})/|c|^2, \qquad \text{(components of } c_t\text{, not } \boldsymbol{c}_t\text{)}. \tag{3.20}$$

Taking $\boldsymbol{c}$ to be $\mathcal{R}$'s centroid gives the kinetic energy

$$T = \tfrac{1}{2} M\,|\,\dot{\boldsymbol{c}}_t|^2 + \tfrac{1}{2}\mathbf{I}_{\boldsymbol{c}}\,\omega^2. \tag{3.21}$$

In the above derivation, many operations look like complex number arithmetic. In particular for $u, v \in \mathbb{R}^2$, $[u, \bigtimes u]\,v$ (matrix-vector product) equals uv (complex number product). The following reformulation of (3.19, 3.20, 3.21) is easily derived.

THEOREM 3.2. *With the notation above, treating vectors as complex numbers:*

(i) $\boldsymbol{c}$*'s WF position* $\boldsymbol{c}_t$ *and velocity* $\dot{\boldsymbol{c}}_t$ *at any time are related to those of* $\boldsymbol{a}$, $\boldsymbol{b}$ *by*

$$a\,\boldsymbol{a}_t + b\,\boldsymbol{b}_t + c\,\boldsymbol{c}_t = 0, \qquad a\,\dot{\boldsymbol{a}}_t + b\,\dot{\boldsymbol{b}}_t + c\,\dot{\boldsymbol{c}}_t = 0. \tag{3.22}$$

(ii) Let $\boldsymbol{c}$ *be* $\mathcal{R}$*'s centroid, then the kinetic energy is*

$$T = \tfrac{1}{2}\left(M\,|a\,\dot{\boldsymbol{a}}_t + b\,\dot{\boldsymbol{b}}_t|^2 + \mathbf{I}_{\boldsymbol{c}}\,|\,\dot{\boldsymbol{b}}_t - \dot{\boldsymbol{a}}_t|^2\right)/|c|^2.$$

A useful case, using the formula $M\ell^2/12$ for a rod's MoI about its centroid, is:

COROLLARY 3.3. *For a uniform thin rod of mass* M*, taking the reference points to be its two ends* $\boldsymbol{a}$, $\boldsymbol{b}$*, its kinetic energy is*

$$T = \frac{M}{6}\left(|\,\dot{\boldsymbol{a}}_t|^2 + \dot{\boldsymbol{a}}_t \cdot \dot{\boldsymbol{b}}_t + |\,\dot{\boldsymbol{b}}_t|^2\right).$$

Here normal vector notation is used, with $\cdot$ *being the dot product.*

```yaml
Title:          Mechanism0
PhysicalParams:
  g:           -9.80665 # gravity
  L:            1       # rod length
  M:            1       # rod mass
  k:            1000    # spring stiffness
  l:            1       # spring rest-length
  mu:           .5      # spring mass
  m:            .1      # particle mass
Dimension:      2
PartData:
  Fixed:        {O: []}
  Rigids:
    OA: {Geom: [[L]], Dyna: [[L/2],M,M*L**2/12]}
  Springs:
    AB:         [ k, l, mu ]
  Particles:
    B:          m
AppliedForces:
  Gravity:
ProblemData:
  t0:  0
  tend: 1
  positions: {A: [[L,fixed],-L],B: [L+l,-(L+l+m*g/k)]}
  velocities: {A: [1,0]}
```

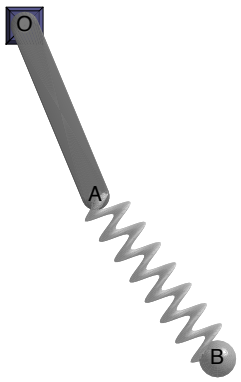


FIG. 2. *Mechanism0. Listing on the left is from file* `mechanism0-p.yaml`*, omitting sections that guide the numerical solution and the animation.*

4. The mechanism facility by examples.

4.1. Overview. As the user sees it, the mechanism facility is a way to specify a "supported" mechanism by a *MechSpec* file, converted automatically to a Lagrangian plus constraints and solved by DAETS. The file—for details see [9]—is written in YAML, a human-readable data serialization language that can handle data of arbitrary structure and content. YAML-readers for various languages—in our case, C++ for the numerics and MATLAB for the animation—convert the text file to an internal program data structure.

For the numerics, the program interprets this to "create" parts, forces and constraints. Currently, the MechSpec only supports 2D mechanisms, and these features:

Parts rigid body (of nonzero size), particle (a point), spring. Each may or may not have mass.

Connection (a) pin-joint, the constraint that two or more RUPs in different frames coincide in the world-frame;
(b) collinear, the constraint that three or more RUPs lie on a straight line, which can implement various kinds of sliding or rotating joint, see §6.1.

Forces (a) terrestrial gravity;
(b) a force at a point fixed on a body, constant in the local frame or WF;
(c) constant torque on a body or between bodies.

The rest of §4 explains the MechSpec via examples.

4.2. Mechanism0. The right of Figure 2 shows Mechanism0. A rod OA of length L and mass M is pivoted at fixed point O; at A is pivoted a spring AB of stiffness constant k, rest-length ℓ and mass μ; at B is fastened a particle of mass m. The system moves under gravity. The parts have "thin rod" and "particle" dynamics despite being drawn as 3D objects. The "spring with mass" is idealised as a stretchable uniform rod, neglecting internal modes of motion a real spring would have.

On the left of Figure 2 is a YAML description of it. Lines 1–20 define the mechanism. Lines 21–25 specify a particular IVP for numerical solution. Names in bold such as **`Title`** are mandatory keywords; others such as `k` or `OA` are defined by the user for a given mechanism.

Basic YAML syntax/semantics. For more detail see [9].

- Whitespace creates indentation, is usually needed after `:`, and is otherwise mostly ignored. From # to end of line is comment.
- **Literal** values for our purpose are either numbers or text-strings. Strings don't need quotes, unless they contain characters such as `:` that would cause ambiguity.
- Literals are formed into larger structures by these constructs:
  set using braces, e.g. `{A,B,C}` is the unordered set containing these elements.
  list using brackets, e.g. `[A,B,C]` is the ordered list containing these elements.
  maplet or key-value pair, e.g. `X: x` with key `X` and value `x`. A **dictionary** is a set of maplets.
- Equivalent to defining a set with braces is putting its items on separate lines without braces, indented. (There is a similar format for lists, which we do not use.) So line 12 could be written

```
OA:
  Geom: [[L]]
  Dyna: [[L/2],M,M*L**2/12]
```

Conversely, the **`PhysicalParams`** dictionary could be written

```
PhysicalParams: {g: -9.80665, k: 1000, l: 1, L: 1, m: .1, M: 1}
```

A YAML reader for MATLAB converts this to a structure—call it `data`—within a program, where each dictionary becomes a `struct` object whose field-names are the keys, each field storing the corresponding value. At the top level the whole file defines a dictionary whose keys are `Title`, `PhysicalParams`, etc., making `data` a struct with these fields. Using dot-notation, for instance

`data.Title` accesses the corresponding value, the string `"Mechanism0"`.

This happens recursively as needed, e.g. the value of key `PhysicalParams` is itself a dictionary, so `data.PhysicalParams.k` accesses the value of `k`, the number 1000.

Nedialkov's team has added a preprocessing stage to evaluate expressions that use **`PhysicalParams`** values and write an intermediate YAML file containing only numeric values, which our programs then read. E.g. on line 14, the expression `M*L**2/12`—a moment of inertia $M\ell^2/12$—is evaluated with $\ell = 1$ and $M = 1$, and written to the output. This feature, which standard YAML does not yet have, gives the convenience of referring to physical parameters symbolically.

MechSpec naming convention. The name of a reference or useful point is a single upper or lower case letter followed by zero or more digits. A part's name is a concatenation of RUPs on it. These rules ensure a part name determines the list of its RUPs and vice versa. RPs must be first in the list; by the frame convention (3.5), their order defines the part's local frame. Otherwise the order is user-defined.

The mechanics. Having got the data into a program, we describe its meaning.

`Fixed` in the **`PartData`** section is a dictionary of coordinates of the RUPs fixed in the world-frame, in this case just point O. Coordinates (x, y) are denoted by a YAML list `[x,y]`. A *padding* convention is used, namely if too few coordinates are given, they are padded with trailing zeros. Thus `[]` means `[0,0]`, meaning O is at $(0, 0)$.

`Rigids` is a dictionary of the rigid parts, here just the rod OA, named by its RUPs. This fixes the local frame: O is at $(0, 0)$ and **`Rigids`**`.OA.`**`Geom`** is a list of local coordinates of the *remaining* RUPs. Using padding, `[L]` is short for `[L,0]` so this says A is at $(L, 0)$.

`Rigids``.OA.`**`Dyna`** lists the coordinates of OA's centroid, its mass and its MoI about the centroid: using padding these are respectively $(L/2, 0)$, M and $M\ell^2/12$.

`Springs` specifies springs similarly. A spring is named by two RPs, its two ends in a user-chosen order: here, there is one spring AB. Its local frame has origin at A and x axis in the direction from A to B. The needed data for a spring is a list giving the stiffness k, the rest-length ℓ, and optionally the mass μ which is zero if omitted.

A spring-length should stay > 0, as if A, B coincide during motion the local frame becomes indeterminate. However when this occurs during numerical solution it usually seems not cause problems.

`Particles` specifies particles similarly. A particle is named by the RP giving its position, and the only data needed is its mass.

In the **`AppliedForces`** section, this mechanism has just (constant) terrestrial gravity. The general form is `Gravity: [`g_x`,`g_y`]` giving it as a vector (g_x, g_y) in the world-frame. Omitting `[`g_x`,`g_y`]`, as here, gives the SI units default value $(0, -9.80665)$, which also declares "upwards" to be the $+y$ direction.

Lagrangian. By convention, the coordinate vector $\mathbf{q}$ comprises all moving reference points, in lexicographic order of their names. Here this means

$$\mathbf{q} = (A, B) = (A_x, A_y, B_x, B_y). \tag{4.1}$$

where A_x is the x-coordinate of A, and so on. From the above data the mechanism facility has the information to construct all the kinetic energy KE and potential energy PE contributions as functions of $\mathbf{q}$ and $\dot{\mathbf{q}}$:

(4.2)

KE	gravitational PE	spring PE
$T_{OA} = \dfrac{M}{6}\lvert\dot{A}\rvert^2,$	$V_{OA} = \dfrac{Mg}{2}A_y,$	
$T_{AB} = \dfrac{\mu}{6}(\lvert\dot{A}\rvert^2 + \dot{A}\cdot\dot{B} + \lvert\dot{B}\rvert^2),$	$V^{[g]}_{AB} = \dfrac{\mu g}{2}(A_y + B_y),$	$V^{[s]}_{AB} = \dfrac{k}{2}\left(\lvert B - A\rvert - \ell\right)^2,$
$T_B = \dfrac{m}{2}\lvert\dot{B}\rvert^2,$	$V_B = mgB_y.$	

The rows refer to rod, spring and particle respectively. Corollary 3.3 was used for the rod and spring KEs. Superscripts $^{[g]}$ and $^{[s]}$ mark the two parts of the spring's PE. Finally, the system is specified by the Lagrangian:

$$\mathcal{L} = \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = T - V = (T_{OA} + T_{AB} + T_B) - (V_{OA} + V^{[g]}_{AB} + V^{[s]}_{AB} + V_B) \tag{4.3}$$

and one constraint, that the rod has fixed length,

$$0 = l_{OA}(\mathbf{q}) = |A|^2 - L^2, \tag{4.4}$$

which are then processed by the Lagrangian facility.

Output file. At each integration time point t during numerical solution, DAETS writes to a file for use by the animation and any other post-processing. The default columns of output are, in order:

- t
- **q**—all the moving points plus any turn angles, see §4.3—interleaved with its first derivative. The ordering is all moving point names and then all turn angles, each in lexicographical order. E.g. for Mechanism0 this comprises $A_x, \dot{A}_x, A_y, \dot{A}_y, B_x, \dot{B}_x, B_y, \dot{B}_y$.
- T, V—the total kinetic and potential energies, useful for checking energy conservation.

This output can be varied in the **`SolverParams`** section of the YAML file.

Signature matrix. An option in the **`SolverParams`**.**`Display`** section of the data file generates the *signature matrix* of the DAE, that is the matrix $\Sigma = (\sigma_{kl})$, where σ_{kl} is the highest order of derivative of the lth variable occurring in the kth equation, or $-\infty$ if it does not occur.

For Mechanism0, with some labels added, it looks as follows. A dash means $-\infty$.

```
TABLEAU
-------
     |  0   1   2   3   4   5   6  |c_i
   ------------------------------------
    0|  2   0   2*  0   0   -   -  | 0    A_x
    1|  0   2   0   2   0*  -   -  | 0    A_y
    2|  2*  0   2   0   -   -   -  | 0    B_x
    3|  0   2   0   2*  -   -   -  | 0    B_y
    4|  0   0*  -   -   -   -   -  | 2    l_OA
    5|  1   1   1   1   -   0*  -  | 0    T
    6|  0   0   0   0   -   -   0* | 0    V
   ------------------------------------
  d_j|  2   2   2   2   0   0   0  |
       A_x  A_y  B_x  B_y  l_OA  T   V
    Index = 3, DOF = 6
```

The labels are the same for rows and columns. For columns they mean the n_q (= 4 by(4.1)) components of **q**, followed by the Lagrange multipliers of the n_c (= 1 by(4.4)) constraint equations, followed by T and V.

For rows they denote the n_q Euler–Lagrange equations generated by the q_j, followed by the n_c constraints, followed by the equations that define T and V. Apart from the T, V rows and columns, the matrix is always symmetric.

This display is a general-purpose feature of DAETS, so not angled to Lagrangian systems. The reported value `Index` is always 3 if $n_c > 0$. The reported value `DOF` is the *total* degrees of freedom, thus we halve it to get DOF as defined in (1.4), namely DOF $= n_q - n_c = 4 - 1 = 3 =$ `DOF`$/2$.

DOF is relevant for the user, since (see below (1.4)) for a well-defined IVP, DOF initial positions and DOF initial velocities must be specified **`fixed`** in the YAML file. Giving more items **`fixed`** is flagged as an error; giving fewer is not, but DAETS may alter non-fixed IVs while finding a consistent (in the DAE sense) initial solution.

The T, V rows may be used as a check of what moving points contribute to the energy. Here, they say T depends on all of $\dot{A}_x, \dot{A}_y, \dot{B}_x, \dot{B}_y$ and V depends on all of A_x, A_y, B_x, B_y,—as one would expect.

4.3. Including torques. Currently the mechanism facility supports constant torque τ between two bodies, one of which may be the WF. When the bodies make a relative turn through an angle θ, the torque does work $\tau\theta$. Thus, it is a conservative force that we can include in the Lagrangian as a potential energy term $V = -\tau\theta$. A $\mathbf{q}$ consisting only of cartesian coordinates cannot safely model this, as it cannot tell θ from $\theta + 2\pi$, so we include the turn angle explicitly as a component of $\mathbf{q}$.

To illustrate, add to Mechanism0 a constant torque τ on rod OA—that is, between OA and the world-frame. To specify this in the data file, add to the **AppliedForces** section the line

```
ConstTorques: {OA: tau}
```

and give `tau` ($= \tau$) a value in the **PhysicalParams** section. The effects are:

1. Append to $\mathbf{q}$ the variable θ_{OA} defined as the angle from the WF x axis to OA's local-frame x axis.[1]
2. Replace the single constraint (4.4) by (in complex number terms) $A{=}Le^{i\theta_{OA}}$, or as two real equations

$$A_x = L\cos\theta_{OA}, \quad A_y = L\sin\theta_{OA}.$$

3. Include in (4.2) and hence (4.3) the potential energy contribution

$$V_{OA}^{[\tau]} = -\tau\theta_{OA}. \tag{4.5}$$

Items 1, 2 keep the balance between number of variables and number of equations by adding 1 to each. The Euler–Lagrange equations then automatically track θ as a continuous quantity.

4.4. Validation and numerical results for Mechanism0. As a check we described Mechanism0 in a Lagrangian form without constraints, leading to an ODE as below. This was transcribed into MATLAB and solved using the MATLAB ODE suite. We found close agreement between this and the mechanism facility solution, and take this as sufficient evidence that the code of both methods, and equations (4.6)–(4.8), 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

$$\left.\begin{aligned} A &= L\,(\sin\theta, -\cos\theta), \quad B = A + r\,(\sin\phi, -\cos\phi), \\ \dot{A} &= L\,(\cos\theta, \sin\theta)\dot{\theta}, \quad \dot{B} = \dot{A} + (\sin\phi, -\cos\phi)\dot{r} + r\,(\cos\phi, \sin\phi)\dot{\phi}. \end{aligned}\right\} \tag{4.6}$$

[1] Made unique by being a continuous function of t, with initial value lying in $(-\pi, \pi]$.

The Lagrangian is still given by (4.3) with extra term (4.5). In the new coordinates (4.2) becomes

$$\left.\begin{array}{lll}
\text{KE} & \text{gravitational PE} & \text{spring or torque PE} \\
T_{OA} = \tfrac{1}{6}ML^2\dot\theta^2, & V_{OA} = -\tfrac{1}{2}MgL\cos\theta & V_{OA}^{[\tau]} = -\tau\theta \\
T_{AB} = \tfrac{1}{2}\mu\Big(L^2\dot\theta^2 + L\sin(\phi-\theta)\dot\theta\dot r + Lr\cos(\phi-\theta)\dot\theta\dot\phi + \tfrac{1}{3}(\dot r^2 + r^2\dot\phi^2)\Big), & & \\
\qquad V_{AB}^{[g]} = -\mu g(L\cos\theta + \tfrac{1}{2}r\cos\phi), & V_{AB}^{[s]} = \tfrac{1}{2}k\,(r-\ell)^2\,, & \\
T_B = \tfrac{1}{2}m\Big(L^2\dot\theta^2 + 2L\sin(\phi-\theta)\dot\theta\dot r + 2Lr\cos(\phi-\theta)\dot\theta\dot\phi + \dot r^2 + r^2\dot\phi^2\Big), & & \\
\qquad V_B = -mg(L\cos\theta + r\cos\phi), & &
\end{array}\right\}$$

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

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\mathbf{q}}} = \frac{\partial\mathcal{L}}{\partial\mathbf{q}}, \text{ call the latter } F(\mathbf{q},\dot{\mathbf{q}}). \tag{4.7}$$

To cast this as a second-order ODE for $\mathbf{q}$ is possible but needlessly complicated. Instead, introduce the Hamiltonian generalised momentum vector

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

Now $\mathcal{L}$ is a quadratic form as a function of $\dot{\mathbf{q}}$, so $\mathbf{p}$ depends linearly on $\dot{\mathbf{q}}$, namely

$$\mathbf{p} = \mathcal{M}(\mathbf{q})\dot{\mathbf{q}},$$

where $\mathcal{M}$ is a matrix (symmetric because it is the Hessian $\partial^2\mathcal{L}/\partial\dot{\mathbf{q}}^2 = \partial^2(T_{OA} + T_{AB} + T_B)/\partial\dot{\mathbf{q}}^2$). Thus we can cast (4.7) as two first-order vector ODEs, making 6 scalar equations:

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

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

$$\mathcal{M}(\theta, r, \phi) = \begin{bmatrix} \left(\frac{1}{3}M + \mu + m\right)L^2 & (\frac{1}{2}\mu + m)L\sin(\phi-\theta) & (\frac{1}{2}\mu + m)Lr\cos(\phi-\theta) \\ (\frac{1}{2}\mu + m)L\sin(\phi-\theta) & \frac{1}{3}\mu + m & 0 \\ (\frac{1}{2}\mu + m)Lr\cos(\phi-\theta) & 0 & (\frac{1}{3}\mu + m)r^2 \end{bmatrix} \tag{4.8}$$

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. This made it simple to compare its output with the mechanism facility's, without a separate interpolation stage.

The mechanism facility program and the MATLAB program read the same YAML problem file. The IVs specify the position and velocity of $A = (A_x, A_y)$ and $B = (B_x, B_y)$. They might not be consistent: A might not lie on, and $\dot A$ 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

```
PhysicalParams: {L: 11.137, M: 2.059, k: 35.24, l: 11.88, m: 1.416, mu: 2.715,
  torque: 6.46, v0: 12.34}
ProblemData:
  t0       :    0
  tend     :    10
  positions : { A: [[L*5/13, fixed], -L*12/13],
                B: [[L*5/13+0.6*l, fixed], [-L*12/13-0.8*l, fixed]] }
  velocities: { A: [[v0, fixed], 0.0], B: [[0.0, fixed], [0.0, fixed]] }
```

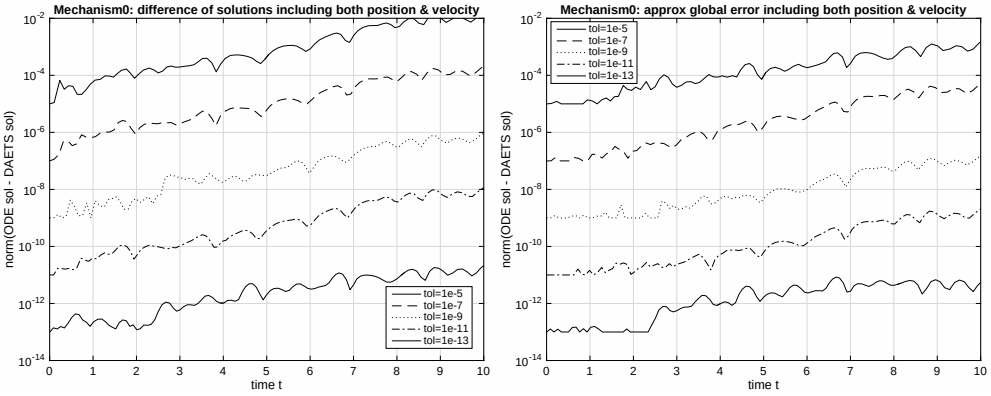


FIG. 3. *Above: Parameters and IVs for Mechanism0 problem. Below left: comparison of mechanism facility and ODE solutions at different tolerances. Below right: estimate of global error, (*DAETS *solution)–(*`ode113` *solution at tightest tolerance).*

fixed and also one of $\dot{A}_x, \dot{A}_y$. 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 $A_y = \pm\sqrt{L^2 - A_x^2}$. We omit details to do with choice of sign and possible numerical ill-conditioning.

For a test of agreement between DAETS and ODE solution, we used MATLAB's `ode113`, an Adams-Moulton code. The top of Figure 3 shows the "slightly random" parameters and IVs chosen. They give mild swinging and spring oscillation, without the chaotic behaviour that is possible. For each tolerance **`tol`**, the 8-element vector $\mathbf{v} = (A, B, \dot{A}, \dot{B})$ (both positions and velocities) was formed for each solver at each t.

The bottom left of the figure 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.

The bottom right of the figure 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 at the smallest tolerance it allows, about `2.2e-14`. 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 DAETS is somewhat more accurate than `ode113`, so the difference $\mathbf{v}_{\text{DAETS}} - \mathbf{v}_{\text{ODE}}$ is mostly due to global error in the `ode113` solutions.

4.5. Mechanism1. The right of Figure 4 shows Mechanism1. It has three identical thin rods AC, BD and EF of length ℓ, and a uniform rigid 45°45°90° triangle CDE of short side ℓ. The parts move under gravity. They pivot without friction at

C, D, E, and at fixed points A, B, where AB is horizontal of length L.

```
1  Title:           Mechanism1
2  Dimension:       2
3  PhysicalParams:
4    l:             1      # ℓ = length of rods
5    L:             0.58   # L = distance AB
6    m:             1      # m = mass of rods
7    M:             1      # M = mass of plate
8  PartData:
9    Fixed:         {A: [-L/2], B: [L/2]}
10   Rigids:
11     AC, BD, EF: {Geom: [[l]], Dyna: [[l/2], m, m*l**2/1
12     CDE:
13       Geom: [[l*sqrt(2)], [l/sqrt(2), -l/sqrt(2)]]
14       Dyna: [[l/sqrt(2), -l/(3*sqrt(2))], M, M*l**2/9]
15 AppliedForces:
16   Gravity:
```

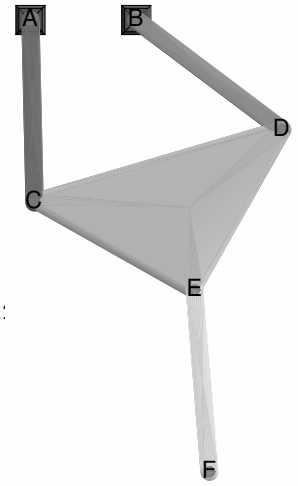


FIG. 4. *Mechanism1. Listing on the left is from* `mechanism1-p.yaml`.

On the left is a YAML description. Line 16 switches on gravity and puts the $+y$ axis upward, so the $+x$ axis is horizontal to the right. In **PartData**, line 9 places A and B at $(\mp L/2, 0)$, making AB horizontal. (Recall the padding convention.) $L = (2 - \sqrt{2})\ell$ is a critical value under which the system can rotate all way round, and above which it cannot. The given $L = 0.58$ is just below this.

PartData defines the vector $\mathbf{q}$, namely all point names occurring in a part are listed, the **Fixed** ones removed, and the result put in lexicographic order to yield

$$\mathbf{q} = (C, D, E, F) = (C_x, C_y, D_x, D_y, E_x, E_y, F_x, F_y).$$

If torques were present, their turn angles would go at the end of this.

As all parts are rigid bodies, **PartData**.**Rigids** gives their geometry and dynamics. Line 11 shows one can specify several identical parts simultaneously (in a comma-separated sequence with optional space). In their local frames, A, B, E are the origins of the three rods, whose other ends C, D, F are at $(\ell, 0)$. Each has centroid at the midpoint, mass m and MoI $m\ell^2/12$. The triangle has C at the local origin, D at $(\ell\sqrt{2}, 0)$, E at $(\frac{\ell}{\sqrt{2}}, -\frac{\ell}{\sqrt{2}})$. By standard formulae its centroid is at $(\frac{\ell}{\sqrt{2}}, \frac{-\ell}{3\sqrt{2}})$ and its MoI about the centroid is $\frac{M\ell^2}{9}$.

Mechanism1 summary. From the above data the mechanism facility constructs the kinetic and potential energy contributions and the Lagrangian $L(\mathbf{q}, \dot{\mathbf{q}})$. These are similar to Mechanism0's (4.2, 4.3) so are omitted. It constructs six constraints—a length constraint for each rigid part, plus the equation from (3.22), equivalent to two scalar constraints, that places the UP E in the correct world position relative to C, D,

as follows.

$$\left.\begin{aligned} l_{AC}: &\quad 0 = C_1(\mathbf{q}) = |C_t - A|^2 - \ell^2, \\ l_{BD}: &\quad 0 = C_2(\mathbf{q}) = |D_t - B|^2 - \ell^2, \\ l_{CDE}: &\quad 0 = C_3(\mathbf{q}) = |D_t - C_t|^2 - 2\ell^2, \\ E_{x,CD}, E_{y,CD}: &\quad 0 = C_4(\mathbf{q}) + iC_5(\mathbf{q}) = cC_t + dD_t + eE_t, \\ l_{EF}: &\quad 0 = C_6(\mathbf{q}) = |F_t - E_t|^2 - \ell^2, \end{aligned}\right\}$$

On the left are labels, e.g. l_{AC} is the length constraint for AC, and $E_{x,CD}$ is the x component of the placement of E relative to C, D. The $_t$ are a reminder of which points are moving. For $C_4 + iC_5$, values are treated as complex numbers as in (3.22), with these constants computed in CDE's local frame:

$$c = E - D = (-1 - i)\ell/\sqrt{2}, \quad d = C - E = (-1 + i)\ell/\sqrt{2}, \quad e = D - C = \ell\sqrt{2}.$$

The constraints follow the order of parts in **PartData** and within that, for each rigid body, one length constraint for its RPs followed by two placement constraints for each of its UPs.

4.6. Numerical results for Mechanism1. For the numerical experiments in this subsection and in §4.8, we use order 17 in DAETS.

We determine (number of) significant correct digits (SCD) defined as follows. Denote by Q a vector with the components of a reference solution at t_{end}, and by Q^{tol} the vector with the corresponding components computed with tolerance tol at t_{end}. Then

$$\text{SCD} = -\log_{10} \max_i \left\{ |Q_i - Q_i^{\text{tol}}|/|Q_i| \right\}.$$

We use tolerances $\text{tol} = 10^{-4}, 10^{-5}, \ldots, 10^{-13}$. In this subsection, reference solutions are computed with $\text{tol} = 10^{-14}$.

We plot CPU time versus SCD and $(E_t - E_0)/\text{tol}$ versus t, where E_t is total energy at time t, and E_0 is initial energy. The platform is 2.2 GHz Intel Core i7 running Mac OS X.

4.6.1. Mechanism1a. We simulate the Mechanism1 specification from Figure 4 with initial guesses for C, D, and F given as

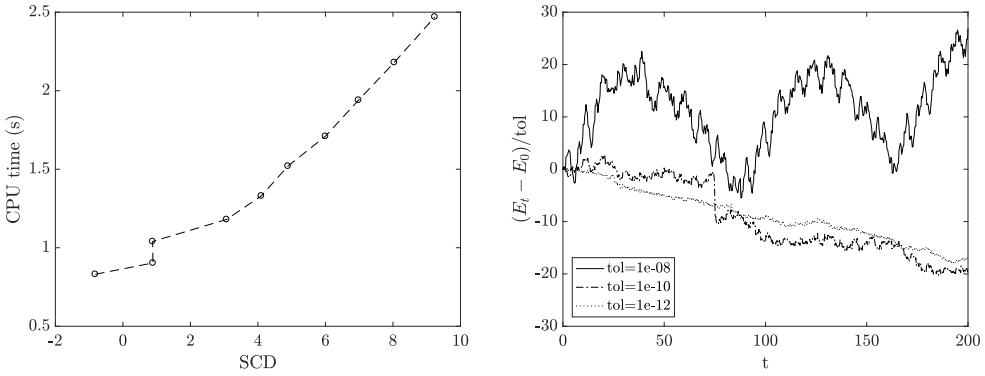
```
positions:   {C:[[-1/sqrt(2),fixed],-1], D:[1/sqrt(2),-1],
      F:[[0,fixed], -1*(1+1/sqrt(2))]}
velocities:  {C:[[2,fixed],0], D:[2,0], F:[[1,fixed],0]}
```

That is, with guesses for positions and velocities

$$\begin{aligned} \big(C_x, C_y;\, D_x, D_y;\, F_x, F_y\big) &= \big(-\ell/\sqrt{2}, -\ell;\, \ell/\sqrt{2}, -\ell;\, 0, -\ell(1 + 1/\sqrt{2})\big) \\ \big(\dot{C}_x, \dot{C}_y;\, \dot{D}_x, \dot{D}_y;\, \dot{F}_x, \dot{F}_y\big) &= \big(2, 0;\, 2, 0;\, 1, 0\big), \end{aligned} \tag{4.9}$$

where C_x, $\dot{C}_x$, F_x, and $\dot{F}_x$ are fixed, and the solver does not change their values when computing a consistent initial point. (The position and velocity of E are determined uniquely from those of C and D.) We refer to Mechanism1 with the initial guesses (4.9) as Mechanism1a.

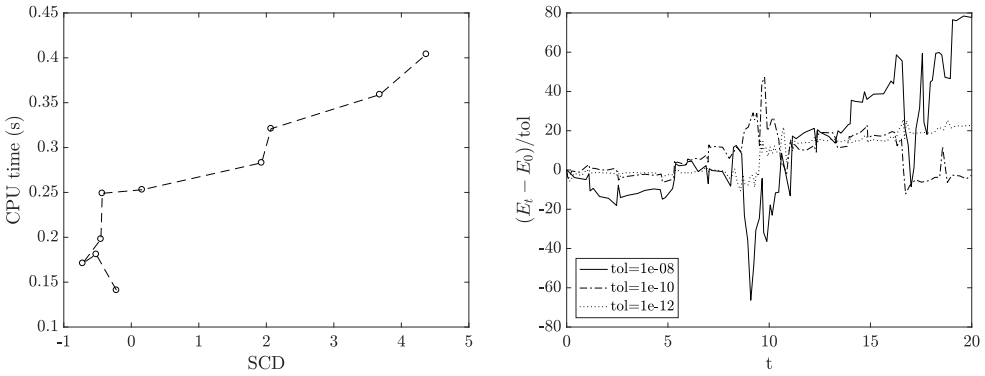
We integrate for $t \in [0, 200]$. The consistent initial values and reference solution found by the solver are given in §A.1. When determining SCD, we include positions and velocities of C, D, and F in Q and Q^{tol}. The corresponding plots are in Figure 5. A discussion of the numerical results in this subsection and in §4.8 is in §4.9.



(a) CPU time vs. SCD (b) energy preservation vs. t

FIG. 5. *Performance of* DAETS *on Mechanism1a.*

4.6.2. Mechanism1b. We repeat the same numerical experiment, but with the masses changed to $m = 2$ and $M = 5$, and the initial values for $\dot{C}_x$ and $\dot{F}_x$ changed to $\dot{C}_x = -4.5$ and $\dot{F}_x = 3$. With such values, this mechanism behaves chaotically, and all SCD are lost at about $t = 30$ and for all tolerances. We integrate to $t_{\text{end}} = 20$, see Figure 6. The consistent initial values at $t = 0$ and the reference solution at $t_{\text{end}} = 20$ are given in §A.2.



(a) CPU time vs. SCD (b) energy preservation vs. t

FIG. 6. *Performance of* DAETS *on Mechanism1b.*

4.7. Andrews Squeezing Mechanism (ASM). Shown in Figure 7, this is problem A03 in the Multi Body Systems (MBS) Benchmark initiated in [14] and implemented under the OpenSim API in [10]. It is also problem II-13 in [5]. From the description in [14, Problem A03]:

> *This plane mechanism is composed by 7 bodies interconnected by revolute joints. The assembly is driven by a motor located at point O, with a constant drive torque of $\tau = 0.033$ Nm. This system has a very small time scale, thus making it difficult to simulate for solvers that can't reach small time-steps.*

Because part OF is driven by a torque, its turn angle from the x axis—θ_{OF} in

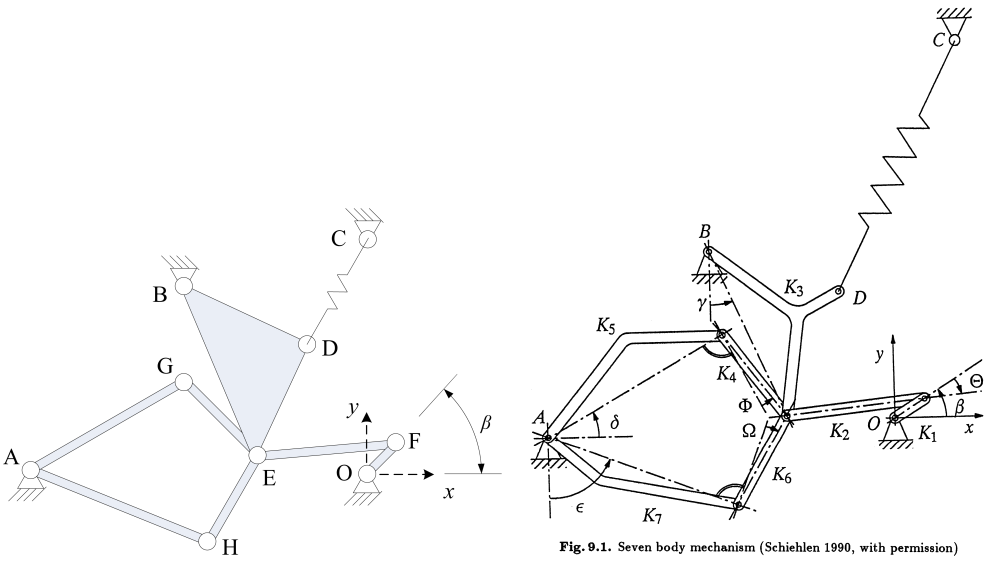


FIG. 7. *Andrews Squeezing Mechanism. (a) As drawn in MSB; (b) original picture in [4, Ch. VII.7].*

the general notation, here called β—must be in the coordinate vector, so

$$\mathbf{q} = (D, E, F, G, H, \beta) = (D_x, D_y, E_x, E_y, F_x, F_y, G_x, G_y, H_x, H_y, \beta).$$

The value of β fixes the position of the whole mechanism. The initial β_0 is given as -0.0620 rad in [10], more accurately in [5, §13.3]. It approximates the unstable equilibrium at which the spring is most stretched, so even a small nonzero torque causes continuous rotation at increasing speed. There is no gravity and no dissipation, so if the system starts from rest, the total kinetic energy at any time is $(\beta - \beta_0)\tau$.

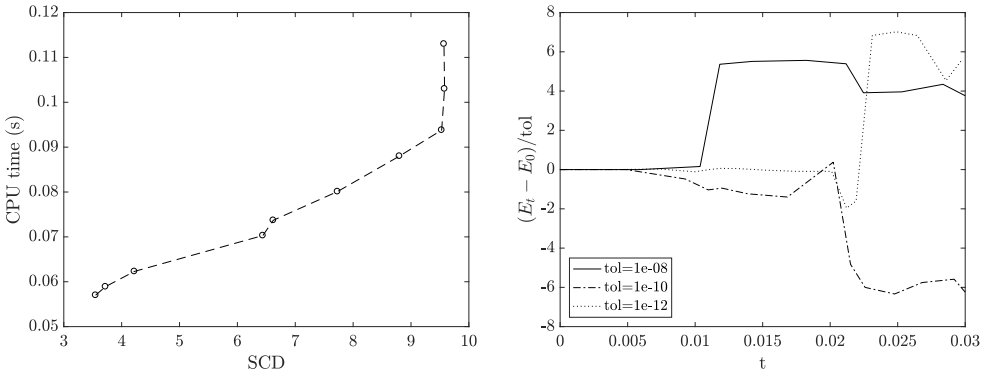
4.8. Numerical results for ASM. This system was solved with the initial condition for β_0 from [5, §13.3] and with the problem parameters as given in [5, §13.3]. See our complete YAML specification in §B. We give initial guesses as

```
positions:
  E: [-2e-02, 1e-03]
  F: [rr*cos(beta0), [rr*sin(beta0),fixed]]
  G: [-3e-02, 1e-02]
  H: [-3e-02, -1e-02]
velocities: # all 0's by default
```

where we have fixed F and given reasonable guesses for E, G, H, else an initial consistent point might not be found. (The picture shows there are 4 configurations for any β, since G and/or H can "flip" to the other side of line AE, though this cannot happen in continuous motion.) The position and velocity of D are determined from those of B and E.

At $t_{\text{end}} = 0.03$ (the value used in [5, §13.3]), we convert our output, which is in cartesian coordinates, except the angle β, to find values for θ, γ, δ, ϵ and Ω, cf. Figure 7(b). Then we use the corresponding reference values from [5, §13.3] to

determine SCD. (In [5, §13.3], derivatives are not included in calculating SCD, and we do not include them either.) The resulting plots are in Figure 7.

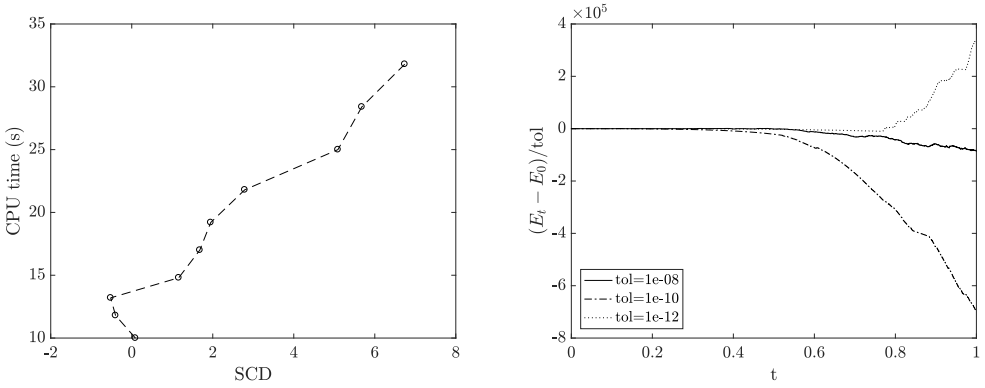


(a) CPU time vs. SCD (b) energy preservation vs. t

FIG. 8. *Performance of* DAETS *on ASM for* $t \in [0, 0.03]$.

We perform a longer integration up to $t_{\text{end}} = 1$ and display the results in Figure 7. In this case, we compute a reference solution with $\text{tol} = 10^{-14}$, see §A.3.

Animations of Mechanisms 1a and 1b and ASM are at [8].



(a) CPU time vs. SCD (b) energy preservation vs. t

FIG. 9. *Performance of* DAETS *on ASM for* $t \in [0, 1]$.

4.9. Discussion. From Figures 5(a), 6(a), 8(a), and 9(a), the CPU time versus SCD grows nearly linearly for tolerances $\approx 10^{-8}$ to $\approx 10^{-12}$. Compared to "hand-coding" of a DAE as e.g. in the test problems of [5], our simulations are generally slower. For example, on the ASM test problem, DAETS's CPU time is about an order of magnitude larger than RADAU's CPU time, see Figure 10. However, our advantage is in much more convenient and easier modelling, which can shorten substantially the time from modelling to obtaining a numerical solution.

Taylor series methods are not symplectic, and they are not expected to preserve the energy well over long integrations, cf. Figure 9 (b). For not very long integrations, however, the energy is preserved reasonably well, cf. Figures 5(b), 6(b), and 8(b).

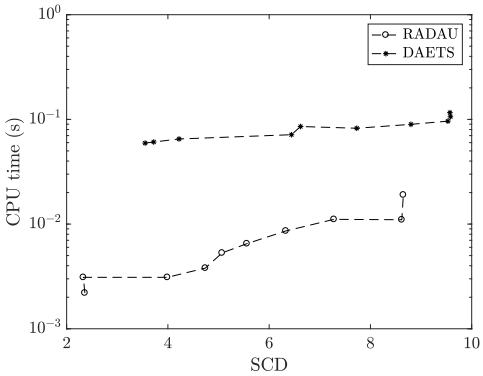


FIG. 10. RADAU *versus* DAETS *on ASM for* $t \in [0, 0.03]$.

5. Animation. Animation can be controlled in the data file **`Animation`** section, of which no more is said here.

Currently we use MATLAB$^{\text{TM}}$ graphics. At each time-point t chosen by its step size control, DAETS has written t and the vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$ to an output file. The animation loop maps these by Hermite cubic interpolation to $\mathbf{q}$ values at a sequence of equally spaced times t_{vid}, and as it does so, produces animation frames in a figure window and optionally sent to a video file. The user can choose real-time frame rate and speed-up/slow-down factor (ratio of model time to real time). The speed of motion in the figure window can be sluggish, but in the video is exact.

Each rigid part (which exists in the xy plane) is given a 2D shape that by default is a round-cornered region slightly larger than the convex hull of its RUPs, and can be customised. This is given a 3D presence by extending it slightly in the z direction to a flat prism, and bevelling the edges. E.g. the ASM by default looks like (a) in Figure 7 but can be made to look like (b), taken from [4, Ch. VII.7], where AG and AH are bent and BED consists of three arms radiating from a central point.

The part is made into a MATLAB `hgtransform` object, containing a 4×4 matrix M specifying its world position. At each t_{vid}, formulae in Section 3 are used (in an inverse direction to the modelling process) to map $\mathbf{q}$ to the part's rotation matrix Q and shift $\boldsymbol{p}$. These are copied into M. (A spring has a more complicated Q, a rotation followed by a scaling along its length.) This instantly puts each part in the correct position in the MATLAB figure, with the benefits of correct hiding of a part by those in front of it, reflective highlights, possible change of view-angle during motion, etc.

6. The 3D case. For 3D the *theory* of tracking points and computing kinetic and where needed potential energy contributions to the Lagrangian is already in §3.1. For the *Lagrangian facility*, the difference between 2D and 3D in the `C++` code amounts mainly to choosing between a `vec2` and a `vec3` vector class and then using the appropriate parts of §3.1. All connections of parts that can be defined in terms of points on bodies being coincident, collinear or coplanar are already covered—see the universal joint example MultPendUJ below. Work needs to be done on specifying applied forces, especially torques in 3D. For the *mechanism facility*, the basic YAML constructs extend in an obvious way to 3D, e.g. how a rigid body's part-name defines its local frame. However we have yet to implement this.

6.1. Other kinds of connection. As in §3, the style A means a (possibly moving) point in fixed space, $\boldsymbol{a}$ its representation as a position vector in some world-

frame, and a a displacement, a difference of position vectors.

With a cartesian approach, many connections between parts reduce to collinearity/coplanarity constraints, hence to simple linear algebra tests. Collinearity is a property of triples of A's and $\boldsymbol{a}$'s but of pairs of a's (a and b are collinear iff linearly dependent). With more than three A's or $\boldsymbol{a}$'s, it means any three of them are collinear, so $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k \in \mathbb{R}^n$ are collinear iff the $(n+1) \times k$ matrix

$$M = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & \ldots & \boldsymbol{a}_k \end{bmatrix}.$$

has rank 1 or 2, and all coincide iff it has rank 1. In order of increasing numerical stability, rank may be found by LU, QR or SVD factorisation of M^T. It is similar for coplanarity, a rank property of quartets of A's and $\boldsymbol{a}$'s but of triples of a's.

Some examples follow, where either body $\mathcal{R}_1, \mathcal{R}_2$ might be fixed space.

- (Any n) A moving rod $\mathcal{R}_1$ slides through a given point C on $\mathcal{R}_2$. Equivalent: A, B, C are collinear where A, B are distinct points on the rod. For $n = 2$ another meaning is the pin-slot joint: $\mathcal{R}_2$ rotates about C which can slide on a line in $\mathcal{R}_1$.
- ($n = 3$) Revolute joint between $\mathcal{R}_1, \mathcal{R}_2$. Equivalent: A, B, C are collinear where A is on both bodies, and AB, AC are the hinge axis lines in $\mathcal{R}_1$, $\mathcal{R}_2$ respectively.
- ($n = 2, 3$) Piston (cylindrical) joint between $\mathcal{R}_1, \mathcal{R}_2$, allowing sliding along an axis and, in 3D, rotation about it. Equivalent: A, B, C, D are collinear where AB, CD are the joint axis lines in $\mathcal{R}_1$, $\mathcal{R}_2$ respectively.
- ($n = 3$) Prismatic joint, allowing sliding along an axis but not rotation. As cylindrical, plus vectors ℓ, a, b are coplanar, where ℓ is along the axis line and a, b are suitable vectors fixed in $\mathcal{R}_1$, $\mathcal{R}_2$ respectively.

6.2. 3D tests done. These examples, done directly using the Lagrangian facility, mainly aim to verify in 3D the QR factorisation approach of §3, and the joint ideas in §6.1. Animations can be viewed at [8].

Skew2Pend. This has two rods with mass, moving under gravity. Rod OA hangs by a hinge (revolute joint) h_1 at one end from fixed point O and at the other by hinge h_2 to rod AB. The hinge axes in both rods are perpendicular to the rod's length, but the system is made "skew" in two ways: (a) in the WF, the axis of h_1 is at an angle α to the horizontal; (b) in OA, the axis of h_2 is rotated through an angle β relative to that of h_1 (as viewed along the rod's length). Here α, β are parameters read as data.

MultPendFP. This, the "free pivot" (or ball-and-socket joint) multi-pendulum, is the example in §1.3 for an arbitrary number N of pendula.

MultPendUJ. In a similar model, we replaced the chain of N particle bobs and massless rods by uniform rods with mass, and the free pivots by universal joints (Cardan or Hookean joints). This joint bends in any direction but can transmit torque, unlike a free pivot. It is usually built as a rigid part $\mathcal{X}$ having two revolute joints h_1, h_2 whose axes are coplanar and at right angles—a "cross". Bodies $\mathcal{R}_1$, $\mathcal{R}_2$ are joined by hinging $\mathcal{R}_1$ to $\mathcal{X}$ at h_1 and $\mathcal{X}$ to $\mathcal{R}_2$ at h_2.

One can choose the hinge axes in $\mathcal{R}_1$ and $\mathcal{R}_2$ at will. For our model, with a chain of rods, we put each hinge perpendicular to the length of its rod, but rotated the hinge at the far end of each rod through a skewing angle α relative to that at the near end, where α is a parameter. It is visually clear (with $N = 10$) that this system moves

"less freely" than the free pivot pendulum. Changing α should change the motion somewhat, but it is hard to spot any difference.

7. Conclusions.

What we have done. We have argued cartesian coordinates generally make Lagrangian modelling of a mechanism simpler than do other kinds of coordinates. This fact is not of great weight as long as the obstacles along this route to simulation are not mainly in *forming* the Lagrangian description, but in *deriving* from it and then *solving* the equations of motion, which are an index-3 DAE.

In reverse order of these three items: there is now a DAE code, DAETS, that *solves* such a DAE as easily as an ODE. At the mathematical level (ignoring possible numerical difficulties, which seem rare for these systems) and for simulation as an IVP (not, e.g., control problems which can change the DAE index), the methods in DAETS provably always succeed on these DAEs.

The *deriving*—often done by a computer algebra system—is streamlined by the Lagrangian facility which does the needed $\partial/\partial q_j$, $\partial/\partial \dot{q}_j$ and $\mathrm{d}/\mathrm{d}t$ in (1.2), invisibly at run time, using algorithmic differentiation. It handles 3D problems as well as 2D.

The *forming* is streamlined, currently for 2D only, by the mechanism facility which converts a text file MechSpec description of a mechanism to a form the Lagrangian facility can use.

As a result, for simple problems one can go in a few seconds from text file to simulation by DAETS, to animation in MATLAB. The numerical results in this paper validate this approach and show its usefulness. For animations online see [8]. The MechSpec format is documented in [9].

Potential benefits. In a constraint-free approach one spends much effort to produce a Lagrangian that tends to be cumbersome and unreadable. The cartesian approach gives a Lagrangian that tends to be short and readable. Now that the equations of motion resulting from this approach can be derived, solved and animated easily, we believe this work can have a valuable impact on education in mechanics.

Namely Lagrangian methods become accessible to mathematicians and engineers at undergraduate level. Once students have mastered some simple examples, they can experiment with changing the parameters and initial conditions, and invent new mechanisms themselves, rapidly gaining physical intuition into such systems.

A disadvantage is that the analytic work that the constraint-free method puts into deriving a coordinate system becomes, in a cartesian approach, the numerical work of finding an *initial consistent solution*: a notorious difficulty for DAEs. DAETS gives this task to the powerful optimisation software IPOPT. However, IPOPT can give wrong results or fail, if given poor initial guesses for the non-**`fixed`** components of **q**: consider the 4 possible initial ASM configurations mentioned in §4.7.

From a pedagogic viewpoint this can be an advantage—one needs to get physical insight into how a set of rigid parts might move before giving numerical IVs, especially for velocities. A model with card and paper fasteners might help! The mechanism facility currently gives little for support for finding suitable IVs: work is needed here.

In fact even for constraint-free angle coordinates, specifying initial conditions needs insight, since often only a certain range of angle values is physically possible.

Future plans. A 3D MechSpec is an important goal. Before moving to it we aim to put in place some features that are useful in both 2D and 3D:

– A general applied force or torque on a body that is a prescribed function of position and velocity.

– A controlled speed facility, e.g. prescribing a rotating part's angular velocity as a function of time.
– A Rayleigh dissipative term, often a convenient way to model damping forces within the Lagrangian.

These have the common feature of user-supplied functions, which for the `C++` main program require a new compile-and-link stage to be added to the current workflow.

The structural analysis built in to DAETS "knows" a lot about IVs. E.g. for the 2-DOF Mechanism1, it knows parts AC, CDE, BD carry one DOF and EF carries the other. That is, subject to constraints noted above, one can specify one of $(C_x, C_y, D_x, D_y, E_x, E_y)$ arbitrarily, which fixes the other five; one can then specify one of (F_x, F_y) arbitrarily and this fixes the other. The same is true for velocities. It will take some work however to make this knowledge conveniently available in the current context.

Finally, a mechanism implemented using this approach is conceptually an acausal set of equations relating variables. At a software architecture level, we aim to encapsulate it as an object, with a suitable interface for making it a component in a larger system such as a control or optimisation problem.

Appendix A. Consistent initial values and reference solutions.

Below are consistent initial values and reference solutions as computed by DAETS. Order is 17, tolerance is 10^{-14}.

A.1. Mechanism1a.

consistent initial values at $t = 0$

C_x	-7.07106781186547462e-01	$\dot{C}_x$	2.00000000000000000e+00
C_y	-9.08857487776932649e-01	$\dot{C}_y$	-9.17870594226585679e-01
D_x	7.07106781186547684e-01	$\dot{D}_x$	1.99999999999999978e+00
D_y	-9.08857487776932538e-01	$\dot{D}_y$	9.17870594226586123e-01
F_x	0.00000000000000000e+00	$\dot{F}_x$	1.00000000000000000e+00
F_y	-2.61596426896347944e+00	$\dot{F}_y$	2.55408735735967164e-16

reference solution at $t = 200$

C_x	-8.58933625820345714e-01	$\dot{C}_x$	1.42392657308349713e-01
C_y	-8.22383444270928132e-01	$\dot{C}_y$	-9.85087569271938662e-02
D_x	5.47952653877345597e-01	$\dot{D}_x$	1.56736017382790666e-01
D_y	-9.66157558764425439e-01	$\dot{D}_y$	4.18466649412351580e-02
F_x	1.20803584735356007e-02	$\dot{F}_x$	5.64120716522558929e+00
F_y	-2.56862038936931958e+00	$\dot{F}_y$	1.30161091919837979e+00

A.2. Mechanism1b.

consistent initial values at $t = 0$

C_x	-7.07106781186547462e-01	$\dot{C}_x$	-4.50000000000000000e+00
C_y	-7.07106781186547573e-01	$\dot{C}_y$	4.49999999999999911e+00
D_x	6.77443189298190029e-01	$\dot{D}_x$	-5.54955604839201833e+00
D_y	-9.95241088811850139e-01	$\dot{D}_y$	-5.43352205433924484e-01
F_x	0.00000000000000000e+00	$\dot{F}_x$	3.00000000000000000e+00
F_y	-2.53074377172524434e+00	$\dot{F}_y$	4.20048794745364340e+00

reference solution at $t = 20$

C_x	8.40061614244513311e-01	$\dot{C}_x$	-7.05205902707705556e-01
C_y	-5.42490999255197281e-01	$\dot{C}_y$	-1.09202993195599807e+00
D_x	1.11525829897918727e+00	$\dot{D}_x$	2.83014496458055076e+00
D_y	8.44688435681409722e-01	$\dot{D}_y$	-1.79339329818557580e+00
F_x	8.30051052373316556e-01	$\dot{F}_x$	3.98217393452508350e+00
F_y	5.54226627139820738e-01	$\dot{F}_y$	1.87729689856974757e+00

A.3. ASM.

consistent initial values at $t = 0$

E_x	-2.09600223463543393e-02	$\dot{E}_x$	0.00000000000000000e+00
E_y	1.29516919370668642e-03	$\dot{E}_y$	0.00000000000000000e+00
F_x	6.98667411545144529e-03	$\dot{F}_x$	0.00000000000000000e+00
F_y	-4.31723064568895491e-04	$\dot{F}_y$	0.00000000000000000e+00
G_x	-3.39972038858399814e-02	$\dot{G}_x$	0.00000000000000000e+00
G_y	1.64619716749976851e-02	$\dot{G}_y$	0.00000000000000000e+00
H_x	-3.16331345074088999e-02	$\dot{H}_x$	0.00000000000000000e+00
H_y	-1.56188686683045412e-02	$\dot{H}_y$	0.00000000000000000e+00
OF	-6.17138900142764485e-02	$\dot{OF}$	0.00000000000000000e+00

reference solution at $t = 1$

E_x	-2.24639386331334338e-02	$\dot{E}_x$	-4.27474460689147762e+01
E_y	6.02483760557903669e-04	$\dot{E}_y$	-1.84761764639941397e+01
F_x	5.24386932234803104e-03	$\dot{F}_x$	-5.43947208190019964e+01
F_y	4.63700706600035217e-03	$\dot{F}_y$	6.15135590135045689e+01
G_x	-3.41833832797379944e-02	$\dot{G}_x$	-4.84668952354823279e+00
G_y	1.68091063937647299e-02	$\dot{G}_y$	8.93087980248283309e+00
H_x	-3.21169018672776974e-02	$\dot{H}_x$	-1.35899912619663397e+01
H_y	-1.69138029692332895e-02	$\dot{H}_y$	-3.45444130482928173e+01
OF	5.98859965509975063e+03	$\dot{OF}$	1.17305667308202155e+04

Appendix B. YAML specification of Andrews Squeezing Mechanism.

The physical parameter names are those used by the description in [4]. For completeness the full list of **SolverParams** is given but most of the values are defaults and could be omitted.

```
Title: Andrews Squeezing Mechanism
PhysicalParams: { beta0: -0.06171389001427644963589484580 01, m1: 0.04325,
  m2: 0.00365, m3: 0.02373, m4: 0.00706, m5: 0.07050, m6: 0.00706,
  m7: 0.05498, xa: -0.06934, ya: -0.00227, xb: -0.03635, yb: .03273,
  xc: .014, yc: .072, c0: 4530, i1: 2.194e-6, i2: 4.410e-7, i3: 5.255e-6,
  i4: 5.667e-7, i5: 1.169e-5, i6: 5.667e-7, i7: 1.912e-5, d: 28e-3,
  da: 115e-4, e: 2e-2, ea: 1421e-5, rr: 7e-3, ra: 92e-5, l0: 7785e-5,
  ss: 35e-3, sa: 1874e-5, sb: 1043e-5, sc: 18e-3, sd: 2e-2, ta: 2308e-5,
  tb: 916e-5, u: 4e-2, ua: 1228e-5, ub: 449e-5, zf: 2e-2, zt: 4e-2,
  fa: 1421e-5, mom: 33e-3 }
Dimension: 2
PartData:
  Fixed: { O: [], A: [xa, ya ], B: [xb, yb], C: [ xc, yc] }
  Rigids:
    OF: { Geom: [ [rr] ], Dyna: [ [ ra ], m1, i1 ] }
    FE: { Geom: [ [d ] ], Dyna: [ [ da ], m2, i2 ] }
    BED: { Geom: [ [ss], [sc, sd] ], Dyna: [ [sa, sb], m3, i3 ] }
```

```
    EG: { Geom: [ [e] ], Dyna: [ [ea ],  m4, i4 ] }
    AG: { Geom: [ [zt] ], Dyna: [ [ta, tb], m5, i5 ] }
    HE: { Geom: [ [zf] ], Dyna: [ [zf-fa], m6, i6 ] }
    AH: { Geom: [ [u] ], Dyna: [ [ua, -ub], m7, i7 ] }
  Springs:
    CD: [ c0, l0 ]
AppliedForces:
  ConstTorques: { OF: mom }
ProblemData:
  t0: 0.0
  tend: 0.03
  positions:
    E: [-2e-02, 1e-03]
    F: [rr*cos(beta0), [rr*sin(beta0),fixed]]
    G: [-3e-02, 1e-02]
    H: [-3e-02, -1e-02]
  velocities: # all 0's by default
SolverParams:
  Mode: solve
  Integration:
    tol: 1e-14
    order: 17
  OutFile:
    tformat: '% .17e'
    qformat: '% .17e'
    points: [ D: 2, E: 2, F: 2, G: 2, H: 2 ]
    angles: [ OF: 1 ]
Animation:
  view: [-5, 27]
  physParamsToShow: [$beta0, $c0, $mom]
  Skeleton:
    zscale: 0.0005
    fleshoutwid: 0.0015
    Skels:
      BED: {path: XBXEXDX, newpts: [ X , [sa, sb] ] }
      AG: {path: YAYGY, newpts: [ Y , [ta, tb] ] }
      AH: {path: ZHZAZ, newpts: [ Z , [ua, -ub] ] }
```

REFERENCES

[1] G. Birkhoff and S. MacLane, *Algebra*, AMS Chelsea Publishing, Providence, RI, third ed., 1999.

[2] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, SIAM, Philadelphia, second ed., 1996.

[3] W. Greiner, *Classical Mechanics: Systems of Particles and Hamiltonian Dynamics*, no. v. 1 in Classical theoretical physics, Springer, 2003.

[4] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems*, Springer Verlag, Berlin, second ed., 1991.

[5] F. Mazzia and F. Iavernaro, *Test set for initial value problem solvers*, Tech. Rep. 40, Department of Mathematics, University of Bari, Italy, 2003. http://pitagora.dm.uniba.it/~testset/.

[6] N. Nedialkov and J. Pryce, *DAETS user guide*, Tech. Rep. 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.

[7] N. S. Nedialkov and J. D. Pryce, *Solving differential-algebraic equations by Taylor series (III): the DAETS code*, JNAIAM J. Numer. Anal. Indust. Appl. Math, 3 (2008), pp. 61–80.

[8] N. S. Nedialkov and J. D. Pryce, *Multi-body Lagrangian simulations*, 2017. YouTube channel, https://www.youtube.com/channel/UCCuLchOx0W0yoNE9KOCYlVQ.

[9] N. S. Nedialkov and J. D. Pryce, *YAML specification of 2D mechanisms for the DAETS Lagrangian facility*, tech. rep., Department of Computing and Software, McMaster University, 2018. In preparation.

[10] OpenSim, *OpenSim implementation of MBS Benchmark*, 2008 (accessed July 2017). http://rehabenggroup.github.io/MBSbenchmarksInOpenSim/index.html.

[11] J. D. Pryce, N. S. Nedialkov, G. Tan, and X. Li, *How AD can help solve differential-algebraic equations*, Optimization Methods and Software, (2018), pp. 1–21. DOI: 10.1080/10556788.2018.1428605.

[12] M. D. Shuster, *A survey of attitude representations*, The Journal of the Astronautical Sciences, 41 (1993), pp. 439–517.

[13] L. Susskind and G. Hrabovsky, *Classical Mechanics: The Theoretical Minimum*, The theoretical minimum, Penguin Books, 2014.

[14] University of Coruña, *Multi body systems (MBS) benchmark*, 2005 (accessed July 2017). http://lim.ii.udc.es/mbsbenchmark.