# Rigid Body Dynamics Notes

SHINJIRO SUEDA, Texas A&M University

This writeup is a step-by-step instruction guide for learning how to write your own rigid body dynamics code. We first use maximal coordinates, where each rigid body is represented with 6 degrees of freedom, with both the angular and translation velocities expressed in axis-aligned body coordinates. Then, we switch to reduced coordinates, where each body is represented not with respect to the world, but with respect to its parent.

## 1 MAXIMAL COORDINATES

Following the notation of Cline and Pai [2003] and Kaufman et al. [2008], we use bold letters, $\boldsymbol{x}$, to denote vectors and points in $\mathbb{R}^3$, and sans serif letters, q, to denote generalized coordinates and related quantities. Everywhere possible, q refers to the generalized coordinates of a body, and $\boldsymbol{x}$ refers to a point on the body in $\mathbb{R}^3$. Time derivatives are indicated with a dot: $\dot{\boldsymbol{x}} \equiv d\boldsymbol{x}/dt$; and material derivatives are indicated with a prime: $\boldsymbol{x}' \equiv d\boldsymbol{x}/du$.

### 1.1 Position Representation

The configuration of a rigid body is represented by the usual $4 \times 4$ transformation matrix consisting of rotational and translational components:

$$ {}^{0}_{i}\mathsf{E} = \begin{pmatrix} {}^{0}_{i}\mathsf{R} & {}^{0}\boldsymbol{p} \\ 0 & 1 \end{pmatrix}. \tag{1.1} $$

The leading subscripts and superscripts indicate that the coordinates of rigid body (or frame) $i$ are defined with respect to the world frame, 0. Thus each column of ${}^{0}_{i}\mathsf{R}$ corresponds to the frame's basis vectors, $\boldsymbol{e}_k$, expressed in world coordinates, and ${}^{0}\boldsymbol{p}$ is the position of the frame's origin expressed in world coordinates. In other words, the first three columns of ${}^{0}_{i}\mathsf{E}$ express the $i^{th}$ frame's x, y, and z axis directions in $0^{th}$ coordinate frame, and the last column of ${}^{0}_{i}\mathsf{E}$ expresses the $i^{th}$ frame's position in the $0^{th}$ coordinate frame. Given a local position ${}^{i}\boldsymbol{x}$ on a rigid body, its world position is

$$ {}^{0}\boldsymbol{x} = {}^{0}_{i}\mathsf{E}\,{}^{i}\boldsymbol{x}, \tag{1.2} $$

where we have omitted the homogeneous coordinates for brevity. Unless otherwise stated, we assume that the reference frame is the world frame, and use a trailing subscript to indicate the frame of a rigid body, as in $\mathsf{E}_i$. With this notation, $\mathsf{E}_i$ transforms a position from the local space of the $i^{th}$ rigid body to world space.

The rotation matrix, R, has the following properties:

$$ \mathsf{R}\mathsf{R}^{\top} = \mathsf{R}^{\top}\mathsf{R} = I, \quad \det(\mathsf{R}) = 1. \tag{1.3} $$

This implies that the columns of R are mutually orthonormal and follow the right-hand rule. Also, the inverse of a rotation matrix is the transpose, which is a very useful property! All $3 \times 3$ matrices with the properties above form a group called the *special orthogonal group in 3 dimensions*, or SO(3). We can write this as $\mathsf{R} \in \mathrm{SO}(3)$.

The group of all $4 \times 4$ transformation matrices of the form Eq. (1.1) is called the *special Euclidean group in 3 dimensions*, or SE(3). We can write this as $\mathsf{E} \in \mathrm{SE}(3)$. Because of its special structure, taking the inverse of E is easy:

$$ \begin{pmatrix} \mathsf{R} & \boldsymbol{p} \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \mathsf{R}^{\top} & -\mathsf{R}^{\top}\boldsymbol{p} \\ 0 & 1 \end{pmatrix} \tag{1.4} $$

Author's address: Shinjiro Sueda, Department of Computer Science & Engineering, Texas A&M University, College Station, TX, sueda@tamu.edu.

The inverse matrix reverses the transformation: ${}^0_i\mathsf{E}^{-1} = {}^i_0\mathsf{E}$. In other words, ${}^0_i\mathsf{E}$ transforms points from $i$ to 0, whereas ${}^0_i\mathsf{E}^{-1}$ transforms points from 0 to $i$.

### 1.2 Velocity Representation

The spatial velocity ${}^i\phi_i$ of a rigid body ${}^0_i\mathsf{E}$ describes the motion of the rigid body at time $t$. The spatial velocity, also called a "twist," is composed of the angular component, ${}^i\boldsymbol{\omega}_i$, and the linear component, ${}^i\boldsymbol{v}_i$, both expressed in body coordinates:[1]

$$ {}^i\phi_i = \begin{pmatrix} {}^i\boldsymbol{\omega}_i \\ {}^i\boldsymbol{v}_i \end{pmatrix}. \tag{1.5} $$

This $6 \times 1$ vector can also be expressed as a $4 \times 4$ matrix similar to the transformation matrix in Eq. (1.1), with the rotational part in the $3 \times 3$ upper-left block and the translational part in the $3 \times 1$ upper-right block.

$$ \left[{}^i\phi_i\right] = \begin{pmatrix} \left[{}^i\boldsymbol{\omega}_i\right] & {}^i\boldsymbol{v}_i \\ 0 & 0 \end{pmatrix}, \tag{1.6} $$

where the $3 \times 3$ matrix, $[\boldsymbol{a}]$, is the cross-product matrix such that $[\boldsymbol{a}]\boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b}$:

$$ [\boldsymbol{a}] = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix}. \tag{1.7} $$

The twist is related to the time derivative of the frame with:

$$ {}^0_i\dot{\mathsf{E}} = {}^0_i\mathsf{E} \begin{pmatrix} \left[{}^i\boldsymbol{\omega}_i\right] & {}^i\boldsymbol{v}_i \\ 0 & 0 \end{pmatrix}. \tag{1.8} $$

The intuition behind Eq. (1.8) is that we must multiply the twist by ${}^0_i\mathsf{E}$ to transform it to world space, since ${}^0_i\dot{\mathsf{E}}$ is in world space whereas ${}^i\phi_i$ is in local space. Again, we sometimes suppress the leading superscript for brevity and write $\phi_i$, assuming that all spatial velocity quantities are expressed in local coordinates. If we simplify Eq. (1.8), we see that the time derivative of the rotation matrix is the upper left $3 \times 3$ portion:

$$ {}^0_i\dot{\mathsf{R}} = {}^0_i\mathsf{R}\left[{}^i\boldsymbol{\omega}_i\right]. \tag{1.9} $$

### 1.3 Logarithms and Exponentials

Recall that:

- A rotation matrix belongs to the special orthogonal group in 3D: $\mathsf{R} \in \mathrm{SO}(3)$.
- A rigid body configuration belongs to the special Euclidean group in 3D: $\mathsf{E} \in \mathrm{SE}(3)$.

For each of these, there exists a corresponding velocity, or "Lie algebra" to be more technically precise:

- Angular velocity belongs to so(3), the Lie algebra of SO(3): $\boldsymbol{\omega} \in \mathrm{so}(3)$.
- Spatial velocity belongs to se(3), the Lie algebra of SE(3): $\phi \in \mathrm{se}(3)$.

As shown in Eq. (1.6) Angular velocity, $\boldsymbol{\omega} \in \mathrm{so}(3)$, can also be expressed as a vector $\boldsymbol{\omega} \in \mathbb{R}^3$ or as a $3 \times 3$ skew symmetric matrix, $[\boldsymbol{\omega}]$. Similarly, spatial velocity, $\phi \in \mathrm{se}(3)$, can also be expressed as a vector $\phi \in \mathbb{R}^6$ or as a $4 \times 4$ matrix, $[\phi]$.

[1] Some authors, such as Murray et al. [1994], put the translational component above the angular component.

We use the matrix logarithm and exponential to go back and forth between SO(3) and so(3), as well as between SE(3) and se(3).

$$\begin{aligned} \mathsf{R} &= \exp([\boldsymbol{\omega}]), \quad [\boldsymbol{\omega}] = \log(\mathsf{R}), \\ \mathsf{E} &= \exp([\phi]), \quad [\phi] = \log(\mathsf{E}). \end{aligned} \tag{1.10}$$

Intuitive interpretation is that if a frame undergoes an angular velocity of $\boldsymbol{\omega}$ for 1 unit of time, then the frame will be rotated by R. Similarly, if a frame undergoes a spatial velocity of $\phi$ for 1 unit of time, then the frame will be transformed by E. For general matrices, these operations can be expensive, but for these types of matrices, there are efficient formulas. See Examples A.11 through A.15 by Murray et al. [1994].



MATLAB example. Here, E is a random SE(3) matrix, and $[\phi] = \log(\mathsf{E})$ is the $4 \times 4$ matrix that represents the "velocity" that transforms the identity matrix into E We can recover E from $\phi$ by taking the exponential: $\mathsf{E} = \exp([\phi])$. Both builtin functions and functions defined in `se3.m` are used.

```
>> E = se3.randE() % Random transformation matrix

E =

   -0.7372   -0.2659    0.6211    0.6877
   -0.1676   -0.8186   -0.5493    0.3886
    0.6545   -0.5091    0.5589    0.9371
         0         0         0    1.0000


>> phibrac = logm(E) % Builtin matrix logarithm

phibrac =

   -0.0000   -2.7242   -0.9257    1.3245
    2.7242   -0.0000   -1.1157   -0.6727
    0.9257    1.1157    0.0000    0.3156
         0         0         0         0


>> phibrac = se3.log(E) % SE(3) matrix logarithm

phibrac =

         0   -2.7242   -0.9257    1.3245
    2.7242         0   -1.1157   -0.6727
    0.9257    1.1157         0    0.3156
         0         0         0         0

>> expm(phibrac) % Builtin matrix exponential

ans =

   -0.7372   -0.2659    0.6211    0.6877
   -0.1676   -0.8186   -0.5493    0.3886
    0.6545   -0.5091    0.5589    0.9371
         0         0         0    1.0000

>> se3.exp(phibrac) % SE(3) matrix logarithm

ans =

   -0.7372   -0.2659    0.6211    0.6877
   -0.1676   -0.8186   -0.5493    0.3886
    0.6545   -0.5091    0.5589    0.9371
         0         0         0    1.0000
```

We can verify Eq. (1.8) with finite differencing. Since this quantity involves SE(3), finite differencing is non-trivial and so we must be careful. $\mathsf{E}$ is a function of time, so we can write

$$\dot{\mathsf{E}} \approx \frac{\mathsf{E}(t+\Delta t) - \mathsf{E}(t)}{\Delta t}, \qquad \mathsf{E}(t+\Delta t) = \mathsf{E}(t)\exp(\Delta t \phi(t)). \tag{1.11}$$

Given some $\mathsf{E}$ and $\phi$, we can verify $\dot{\mathsf{E}}$ as above using $\Delta t$ = 1e-8.

```
>> E = se3.randE()

E =

   -0.9227   -0.3845   -0.0274   -0.5020
    0.2865   -0.6364   -0.7162    0.4564
    0.2579   -0.6687    0.6973   -1.5617
         0         0         0    1.0000

>> phi = randn(6,1)

phi =

   -0.1285
    1.2666
    0.8058
   -0.6903
   -0.5538
   -0.1246

>> dt = 1e-8;
>> E1 = E*se3.exp(dt*phi)

E1 =

   -0.9227   -0.3845   -0.0274   -0.5020
    0.2865   -0.6364   -0.7162    0.4564
    0.2579   -0.6687    0.6973   -1.5617
         0         0         0    1.0000

>> (E1-E)/dt % Finite difference

ans =

   -0.2751    0.7470   -1.2181    0.8533
    0.3944   -0.1388    0.2811    0.2439
   -1.4221   -0.2974    0.2408    0.1054
         0         0         0         0

>> E*se3.brac(phi) % Analytical derivative

ans =

   -0.2751    0.7470   -1.2181    0.8533
    0.3944   -0.1388    0.2811    0.2439
   -1.4221   -0.2974    0.2408    0.1054
         0         0         0         0
```



### 1.4 Material Jacobian

If a rigid body is moving with spatial velocity, $\phi_i$, the world velocity of a point, ${}^i\boldsymbol{x}$, affixed to the rigid body is computed as: ${}^0\dot{\boldsymbol{x}} = \mathsf{J}\phi_i$. More spefically,

$${}^0\dot{\boldsymbol{x}} = \underbrace{\mathsf{R}_i \overbrace{\begin{pmatrix} [{}^i\boldsymbol{x}]^\top & I \end{pmatrix}}^{\Gamma \in \mathbb{R}^{3\times 6}}}_{\mathsf{J}\in\mathbb{R}^{3\times 6}} \phi_i, \tag{1.12}$$

where the $3 \times 6$ matrix, $\mathsf{J}$, the material Jacobian, transforms the local spatial velocity of the rigid body, $\phi_i$, into the velocity of a local point on the rigid body in world coordinates, ${}^0\dot{\boldsymbol{x}}$. Its transpose, a $6 \times 3$ matrix, transforms a point force in world space, $\boldsymbol{f}_0$ into a local wrench acting on the rigid body.

$$\mathsf{f}_i = \mathsf{J}^\top \boldsymbol{f}_0. \tag{1.13}$$

This "transpose" relationship works in a variety of generalized coordinate settings. If there is a matrix that maps generalized velocities into world velocities, then its transpose will map forces in world coordinates back to generalized forces.

MATLAB example

```
>> E = eye(4);
>> phi = [0 0 0 1 0 0]'; % no rotation, x translation
>> R = E(1:3,1:3);
>> xlocal = [1 0 0]'; % Local point at (1 0 0)
>> G = se3.Gamma(xlocal)

G =

     0     0     0     1     0     0
     0     0     1     0     1     0
     0    -1     0     0     0     1

>> J = R*G;
>> vworld = J*phi % World velocity of the local point

vworld =

     1
     0
     0

>> phi = [0 0 1 0 0 0]'; % Z rotation, no translation
>> vworld = J*phi % World velocity of the local point

vworld =

     0
     1
     0
```



## 1.5 Adjoint

Just like how 3D points and vectors must be in the same coordinate space before they can be added (and dotted, crossed, etc.), spatial velocities must also be in the same coordinate space. The spatial velocity transforms from one frame to another according to the adjoint of the coordinate transform, which is defined from the rigid transform ${}^0_i\mathrm{E}$.

$$ {}^0_i\mathrm{Ad} = \begin{pmatrix} {}^0_i\mathrm{R} & 0 \\ [{}^0_i\boldsymbol{p}]\,{}^0_i\mathrm{R} & {}^0_i\mathrm{R} \end{pmatrix}. \tag{1.14} $$

The spatial velocity of the $i^{th}$ rigid body in world coordinates is then

$$ {}^0\phi_i = {}^0_i\mathrm{Ad}\ {}^i\phi_i, \tag{1.15} $$

which is the spatial velocity of the $i^{th}$ rigid body with respect to the world, now expressed in world coordinates.

Let's look at the time derivative of the adjoint, which we'll need to derivate the equations of motion. Dropping the superscripts and subscripts for brevity, we have, from Eq. (1.14),

$$ \dot{\mathrm{Ad}} = \begin{pmatrix} \dot{\mathrm{R}} & 0 \\ [\dot{\boldsymbol{p}}]\mathrm{R} + [\boldsymbol{p}]\dot{\mathrm{R}} & \dot{\mathrm{R}} \end{pmatrix}. \tag{1.16} $$

Looking at the rotational component of Eq. (1.8) gives us $\dot{\mathrm{R}} = \mathrm{R}[\omega]$. The point derivative, $[\dot{\boldsymbol{p}}]$, is a little trickier. (Note $[\dot{\boldsymbol{p}}] \neq [\boldsymbol{v}]$.) Instead, note that $\dot{\boldsymbol{p}}$ is the velocity of the frame origin expressed in world coordinates. So, $[\dot{\boldsymbol{p}}] = [\mathrm{R}\boldsymbol{v}]$, since $\boldsymbol{v}$ is expressed in local coordinates, and R rotates a vector from local to world coordinates. But $[\mathrm{R}\boldsymbol{v}] = \mathrm{R}[\boldsymbol{v}]\mathrm{R}^\top$, because

for an arbitrary $\boldsymbol{x}$,[2]

$$\begin{aligned}[\mathsf{R}\boldsymbol{\nu}](\mathsf{R}\boldsymbol{x}) &= (\mathsf{R}\boldsymbol{\nu}) \times (\mathsf{R}\boldsymbol{x}) \\ &= \mathsf{R}(\boldsymbol{\nu} \times \boldsymbol{x}) \\ &= \mathsf{R}[\boldsymbol{\nu}]\boldsymbol{x} \\ &= (\mathsf{R}[\boldsymbol{\nu}]\mathsf{R}^\top)(\mathsf{R}\boldsymbol{x}),\end{aligned} \tag{1.17}$$

and so $[\dot{\boldsymbol{p}}]\mathsf{R} = [\mathsf{R}\boldsymbol{\nu}]\mathsf{R} = \mathsf{R}[\boldsymbol{\nu}]$. So the final form of the time derivative of the adjoint is

$$\dot{\mathsf{Ad}} = \begin{pmatrix} \mathsf{R}[\boldsymbol{\omega}] & 0 \\ \mathsf{R}[\boldsymbol{\nu}] + [\boldsymbol{p}]\mathsf{R}[\boldsymbol{\omega}] & \mathsf{R}[\boldsymbol{\omega}] \end{pmatrix}. \tag{1.18}$$

This can be factored into a product of two matrices:

$$\dot{\mathsf{Ad}}(\mathsf{E}, \phi) = \underbrace{\begin{pmatrix} \mathsf{R} & 0 \\ [\boldsymbol{p}]\mathsf{R} & \mathsf{R} \end{pmatrix}}_{\mathsf{Ad}(\mathsf{E})} \underbrace{\begin{pmatrix} [\boldsymbol{\omega}] & 0 \\ [\boldsymbol{\nu}] & [\boldsymbol{\omega}] \end{pmatrix}}_{\mathsf{ad}(\phi)}, \tag{1.19}$$

where we have added the parameter list to more be explicit. The second factor, $\mathsf{ad} = \mathsf{Ad}^{-1}\dot{\mathsf{Ad}}$, is the spatial cross product matrix, which is the adjoint action of the Lie algebra on itself [Kim 2012; Selig 2004].

We can verify the equation above with finite differencing. Since this quantity involves SE(3), finite differencing is non-trivial and so we must be careful. Since the adjoint is a function of time, we can write

$$\dot{\mathsf{Ad}} \approx \frac{\mathsf{Ad}(t+h) - \mathsf{Ad}(t)}{h}, \qquad \mathsf{Ad}(t) = \mathsf{Ad}(\mathsf{E}(t)), \qquad \mathsf{Ad}(t+h) = \mathsf{Ad}(\mathsf{E}(t)\exp(h\phi(t))). \tag{1.20}$$

Given some $\mathsf{E}$ and $\phi$, we can verify $\dot{\mathsf{Ad}}$ as above using $h$ = 1e-8.

### 1.6 Equations of Motion

The Newton-Euler equations of motion of a rigid body can be written in a compact form as

$$\begin{aligned}\mathsf{M}_i\dot{\phi}_i &= [\text{Coriolis forces}] + [\text{body forces (e.g., gravity)}] \\ &= \mathsf{ad}(\phi_i)^\top \mathsf{M}_i\phi_i + f_{\text{body}}(\mathsf{E}_i).\end{aligned} \tag{1.21}$$

Here, $\mathsf{M}_i$ is the spatial inertia of the rigid body, and $\mathsf{ad}(\phi_i)$ is the spatial cross product matrix from Eq. (1.19). If gravity is the only force involved, then the body force is

$$f_{\text{body}}(\mathsf{E}_i) = \begin{pmatrix} 0 \\ \mathsf{R}_i^\top m\boldsymbol{g} \end{pmatrix}, \tag{1.22}$$

where $m$ is the linear mass and $\boldsymbol{g}$ is the gravity vector in world coordinates. The top zero indicates that gravity does not affect the angular velocity. For the translational velocity, the multiplication by the transpose of the rotation matrix transforms the gravity direction into body coordinates.

[2] https://math.stackexchange.com/questions/2418256/property-of-skew-symmetric-matrices-of-vectors-multiplied-by-rotation-matrices

Eq. (1.21) can be rearranged to take on the more familiar form as given by Murray et al. [1994] in their Equation (4.16). Let $\mathcal{I}$ be the rotational inertia, and $mI$ be the translational inertia. Then

$$\begin{aligned}
\begin{pmatrix} \mathcal{I} & 0 \\ 0 & mI \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{\omega}} \\ \dot{\boldsymbol{v}} \end{pmatrix} &= \begin{pmatrix} [\boldsymbol{\omega}]^\top & [\boldsymbol{v}]^\top \\ 0 & [\boldsymbol{\omega}]^\top \end{pmatrix} \begin{pmatrix} \mathcal{I} & 0 \\ 0 & mI \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \\
&= \begin{pmatrix} [\boldsymbol{\omega}]^\top & [\boldsymbol{v}]^\top \\ 0 & [\boldsymbol{\omega}]^\top \end{pmatrix} \begin{pmatrix} \mathcal{I}\boldsymbol{\omega} \\ m\boldsymbol{v} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \\
&= \begin{pmatrix} [\boldsymbol{\omega}]^\top \mathcal{I}\boldsymbol{\omega} + [\boldsymbol{v}]^\top m\boldsymbol{v} \\ [\boldsymbol{\omega}]^\top m\boldsymbol{v} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \\
&= \begin{pmatrix} [\boldsymbol{\omega}]^\top \mathcal{I}\boldsymbol{\omega} \\ [\boldsymbol{\omega}]^\top m\boldsymbol{v} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \qquad \text{// since } [\boldsymbol{v}]^\top \boldsymbol{v} = -\boldsymbol{v} \times \boldsymbol{v} = 0 \\
&= -\begin{pmatrix} \boldsymbol{\omega} \times \mathcal{I}\boldsymbol{\omega} \\ \boldsymbol{\omega} \times m\boldsymbol{v} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix},
\end{aligned} \tag{1.23}$$

which is the same as the Newton-Euler equation in body coordinates, as given by Murray et al. [1994] in their Equation (4.16).

## 1.7 Maximal Inertia

Expressing the spatial velocity of a rigid body in local coordinates is advantageous in that the mass matrix is diagonal and can be precomputed at the beginning of the simulation. Wikipedia's article on "List of moments of inertia" is a good reference for some common shapes.[3] For a triangular mesh, we can compute the body-centered frame and its associated mass matrix using volume integration [Mirtich 1996].

For example, the $6 \times 6$ mass matrix of a cuboid whose side lengths are $(\Delta x, \Delta y, \Delta z)$ is

$$\mathsf{M} = \begin{pmatrix} \frac{m}{12}\left(\Delta y^2 + \Delta z^2\right) & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{m}{12}\left(\Delta z^2 + \Delta x^2\right) & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{m}{12}\left(\Delta x^2 + \Delta y^2\right) & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & m \end{pmatrix}, \tag{1.24}$$

where $m = \rho \Delta x \Delta y \Delta z$ is the total mass of the cuboid with density $\rho$.

*1.7.1 Composite rigid bodies.* We can easily combine multiple rigid bodies into a single composite rigid body. This new rigid body will have its own inertia matrix and local (body) coordinate space. Let a composite rigid body be composed of $n$ rigid bodies. The composite rigid body's total mass is simply the sum of the individual mass values. A necessary condition for getting a diagonal inertia matrix is that the origin of the composite rigid body's local frame must be at the mass-weighted average:

$$m_c = \sum_{k=1}^{n} m_k, \quad \boldsymbol{p}_c = \sum_{k=1}^{n} \frac{m_k}{m_c} \boldsymbol{p}_k. \tag{1.25}$$

[3] https://en.wikipedia.org/wiki/List_of_moments_of_inertia

Let $\mathsf{E}_c$ be an axis-aligned frame with $\boldsymbol{p}_c$ as its origin. We can sum up the individual inertia matrices by first transforming them to $\mathsf{E}_c$.

$$\mathsf{M}_c = \sum_{k=1}^{n} {}^{k}_{c}\mathrm{Ad}^{\top}\, \mathsf{M}_k\, {}^{k}_{c}\mathrm{Ad}, \tag{1.26}$$

The resulting inertia matrix is not diagonal—the top-left $3 \times 3$ portion, corresponding to the rotations, is a dense matrix. We can diagonalize this with the eigenvalue decomposition. Let $J$ be the $3 \times 3$ matrix. Then `[V,D] = eig(J)`, and then the eigenvalues in `D` contains the diagonalized inertia entries. The eigen vector matrix, `V`, might be a left-handed matrix, which can be checked by making sure the determinant is +1, or by dotting the 3rd column by the cross product of 1st and 2nd columns. The right-handed eigen vector matrix is then the rotational portion of $\mathsf{E}_c$.

### 1.8 Euler Integration with Maximal Coordinates

For now, we will work with the simplest integration method. If we discretize the acceleration as

$$\dot{\phi} = \frac{\phi^{(k+1)} - \phi^{(k)}}{h}, \tag{1.27}$$

where $h = t^{(k+1)} - t^{(k)}$ is the time step size, then we can rewrite Eq. (1.21) to be at the velocity level at time $t^{(k)}$.

$$\mathsf{M}_i \phi_i^{(k+1)} = \mathsf{M}_i \phi_i^{(k)} + h \left( \mathrm{ad}(\phi_i^{(k)})^{\top} \mathsf{M}_i \phi_i^{(k)} + \mathsf{f}_{\mathrm{body}}(\mathsf{E}_i^{(k)}) \right), \tag{1.28}$$

which can be solved for the new velocities, $\phi_i^{(k+1)}$.

The rigid body configuration $\mathsf{E}_i^{(k+1)}$ can be obtained by integrating $\phi_i^{(k+1)}$. We must be careful here, because $\mathsf{E}_i$ belongs to a non-Euclidean space (SE(3), the Special Euclidean group in 3 dimensions). We use a first order implicit discretization, with the time step $h$:

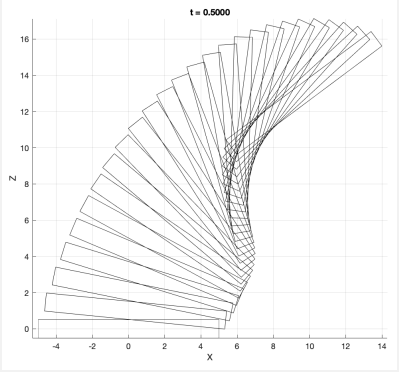
$$\mathsf{E}_i^{(k+1)} = \mathsf{E}_i^{(k)} \exp\left( h \begin{pmatrix} [\boldsymbol{\omega}_i^{(k+1)}] & \boldsymbol{\nu}_i^{(k+1)} \\ 0 & 0 \end{pmatrix} \right). \tag{1.29}$$

The matrix exponential can be computed efficiently using Rodrigues' formula [Murray et al. 1994].

We can finally write the first rigid body dynamics simulator. Run the sample simulation code, `testRigid.m`:

```
>> testRigid
rigid1: 10.00 g
```

Try changing the initial velocity. In the figure to the right, the initial translational velocity is positive in X an Z, while the initial angular velocity is positive in Y.



### 1.9 Elastic and Damping Forces

Following the approach of Baraff and Witkin [1998], we can add implicit damping and elastic forces using the linearly implicit Euler integration. They linearize the forces about the current position and velocity, and move the resulting matrices to the left-hand-side (see §2.5):

$$\left(\mathsf{M} + h\mathsf{D} - h^2\mathsf{K}\right)\phi^{(k+1)} = \mathsf{M}\phi^{(k)} + h\mathsf{f}, \tag{1.30}$$

where D is the damping matrix, K is the stiffness matrix, and f is the sum of all forces. What goes into D, K, and f depends on the type of forces involved.

*1.9.1 Damping Force.* With simple viscous damping, $\mathsf{D} = dI$ is a diagonal matrix, where $d$ is the damping coefficient. There is no contribution to the right-hand-side force vector, f.

*1.9.2 Directional Point Force.* Let's say that we want to pull on a point ${}^i\boldsymbol{x}$ on a rigid body in a particular direction ${}^0\boldsymbol{a}$. (${}^i\boldsymbol{x}$ is in local coords, and ${}^0\boldsymbol{a}$ is in world coords.) Then the linear wrench to be applied to the rigid body can be computed as follows:

$$\mathsf{f} = k\Gamma^\top \mathsf{R}^\top {}^0\boldsymbol{a}, \tag{1.31}$$

where $k$ is the stiffness constant, $\Gamma = ([{}^i\boldsymbol{x}]^\top \; I)$ transforms twists to local point velocities (Eq. (1.12)), and R is the rotation matrix of the rigid body. The corresponding potential energy is

$$V = -k\,{}^0\boldsymbol{x}^\top {}^0\boldsymbol{a}, \tag{1.32}$$

where ${}^0\boldsymbol{x}$ is the position of the force application point in world coordinates. The force in Eq. (1.31) is the negative gradient of this potential energy with respect to the 6 rigid degrees of freedom. We obtain the stiffness matrix if we differentiate again:

$$\mathsf{K} = k\begin{pmatrix}[{}^i\boldsymbol{x}][\mathsf{R}^\top {}^0\boldsymbol{a}] & 0\\ [\mathsf{R}^\top {}^0\boldsymbol{a}] & 0\end{pmatrix}, \tag{1.33}$$

where we used the following identity for the derivatives with respect to the 6 rigid DOFs:

$$\frac{\partial \mathsf{R}^\top \boldsymbol{a}}{\partial \boldsymbol{\omega}} = [\mathsf{R}^\top \boldsymbol{a}], \quad \frac{\partial \mathsf{R}\boldsymbol{a}}{\partial \boldsymbol{\omega}} = -\mathsf{R}[\boldsymbol{a}], \quad \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{\nu}} = \mathsf{R}. \tag{1.34}$$

The stiffness matrix is non-symmetric, which makes sense, since the force is not a function of the rigid translations ($\boldsymbol{\nu}$), and so the second column is zero. We follow Baraff and Witkin [1998] and symmetrize: $\mathsf{K} = \frac{1}{2}(\mathsf{K} + \mathsf{K}^\top)$.

*1.9.3 Point-to-Point Force.* For a linear force between two points on two different bodies, the wrenches acting on these two bodies are

$$\mathsf{f} = k\begin{pmatrix}\Gamma_1^\top \mathsf{R}_1^\top \Delta\boldsymbol{x}\\ -\Gamma_2^\top \mathsf{R}_2^\top \Delta\boldsymbol{x}\end{pmatrix}, \tag{1.35}$$

where $\Delta\boldsymbol{x} = {}^0\boldsymbol{x}_2 - {}^0\boldsymbol{x}_1$, and ${}^0\boldsymbol{x}_1$ and ${}^0\boldsymbol{x}_2$ are the world coordinate positions of the two points, which are obtained by transforming the corresponding local coordinate positions. This force is the negative gradient of the potential energy:

$$V = \frac{1}{2}k\Delta\boldsymbol{x}^\top\Delta\boldsymbol{x}. \tag{1.36}$$

As before, we obtain the stiffness matrix by differentiating the force with respect to the DOFs:

$$\mathsf{K} = k \begin{pmatrix} [{}^1\boldsymbol{x}_1][\mathsf{R}_1^\top(\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & [{}^1\boldsymbol{x}_1] & [{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top\mathsf{R}_2[{}^2\boldsymbol{x}_2] & -[{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top\mathsf{R}_2 \\ [\mathsf{R}_1^\top(\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & I & \mathsf{R}_1^\top\mathsf{R}_2[{}^2\boldsymbol{x}_2] & -\mathsf{R}_1^\top\mathsf{R}_2 \\ [{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top\mathsf{R}_1[{}^1\boldsymbol{x}_1] & -[{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top\mathsf{R}_1 & [{}^2\boldsymbol{x}_2][\mathsf{R}_2^\top(\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & [{}^2\boldsymbol{x}_2] \\ \mathsf{R}_2^\top\mathsf{R}_1[{}^1\boldsymbol{x}_1] & -\mathsf{R}_2^\top\mathsf{R}_1 & [\mathsf{R}_2^\top(\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & I \end{pmatrix}. \tag{1.37}$$

Again, we symmetrize this: $\mathsf{K} = \frac{1}{2}(\mathsf{K} + \mathsf{K}^\top)$.

*1.9.4 Muscle Force.* We use a muscle dynamics model from the biomechanics literature [Millard et al. 2013]. For now, we will assume that we have inextensible tendons, which are called "rigid" tendons in biomechanics, and that the pennation angle is zero. The scalar tension force produced by a muscle is defined to be

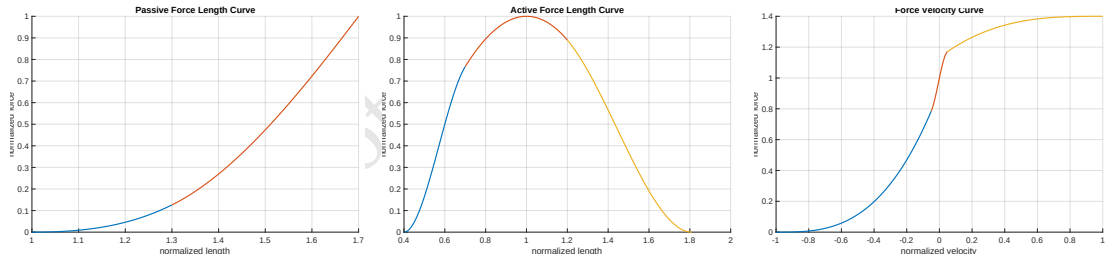
$$f = f_{\text{opt}}\left(a f_{\text{L}}(\tilde{l}) f_{\text{V}}(\tilde{v}) + f_{\text{P}}(\tilde{l})\right), \tag{1.38}$$

where $f_{\text{opt}}$ is the peak isometric force, $a$ is the activation level, $f_{\text{L}}$, $f_{\text{V}}$, and $f_{\text{P}}$ are the active force length, active force velocity, and passive force length curves, which are described below. Once this scalar force is computed, the final generalized forces acting on the two points on the rigid bodies are

$$\mathsf{f} = \frac{f}{\|\Delta\boldsymbol{x}\|}\begin{pmatrix} \Gamma_1^\top\mathsf{R}_1^\top\Delta\boldsymbol{x} \\ -\Gamma_2^\top\mathsf{R}_2^\top\Delta\boldsymbol{x} \end{pmatrix}, \tag{1.39}$$

which is the same as the point-to-point force from before, but with the constant stiffness factor replaced by the new muscle scalar force. The division by $\|\Delta\boldsymbol{x}\|$ normalizes $\Delta\boldsymbol{x}$ in the expression to produce a unit direction for the force to be applied in.

The active force length, active force velocity, and passive force length curves are functions the define some important properties of the muscle. For now, we'll use a simplified versions of these curves, but we can always swap in the more computationally expensive, full-fledged models.




(a) Passive Force Length Curve (b) Active Force Length Curve (c) Force Velocity Curve

Fig. 1. (a) Passive force length curve, composed of two cubic segments. (b) Active force length curve, composed of three cubic segments. (c) Force velocity curve, composed of three cubic segments.

Each curve is composed of a sequence of cubic segments (*i.e.,* piece-wise cubics):

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad f_i'(x) = 3a_i x^2 + 2b_i x + c_i, \quad f_i''(x) = 6a_i x + 2b_i. \tag{1.40}$$

By specifying some conditions on these cubics, we can solve for the coefficients, $a_i$, $b_i$, $c_i$, and $d_i$.

As an example, let's look at the active force length curve, shown in Fig. 1a. Here, we have two cubics: $f_1(x) : x \in [1.0, 1.3]$ and $f_2(x) : x \in [1.3, 1.7]$. So we have 8 coefficients to solve for, and so we need 8 conditions. We want the left cubic to go through (1.0, 0.0) and (1.3, 0.126110265892917)[4], which can be encoded as $f_1(x) = y$ (first two rows of the matrix). We want the right cubic to go through (1.8, 1.0), which can be encoded as $f_2(x) = y$ (row 3). We want the derivative of the left cubic to be 0.0 at $x = 1.0$ and 1.07051016402631 at $x = 1.3$, which can be encoded as $f_1'(x) = dy$ (rows 4 and 5). We want the derivative of the right cubic to be 2.85714285714286 at $x = 1.7$, which can be encoded as $f_2'(x) = dy$ (row 6). Finally, we want the left and right segments to share the same function and derivative at $x = 1.0$, which can be encoded as $f_1(1.3) - f_2(1.3) = 0$ and $f_1'(1.3) - f_2'(1.3) = 0$ (rows 7 and 8). This gives us a square matrix that can be solved for the 8 coefficients of the cubics.

$$\begin{pmatrix} 1.0^3 & 1.0^2 & 1.0 & 1 & 0 & 0 & 0 & 0 \\ 1.3^3 & 1.3^2 & 1.3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.8^3 & 1.8^2 & 1.8 & 1 \\ 3(1.0^2) & 2(1.0) & 1 & 0 & 0 & 0 & 0 & 0 \\ 3(1.3^2) & 2(1.3) & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3(1.7^2) & 2(1.7) & 1 & 0 \\ 1.3^3 & 1.3^2 & 1.3 & 1 & -1.3^3 & -1.3^2 & -1.3 & -1 \\ 3(1.3^2) & 2(1.3) & 1 & 0 & -3(1.3^2) & -2(1.3) & -1 & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{pmatrix} = \begin{pmatrix} 0.0 \\ 0.126110265892917 \\ 1.0 \\ 0.0 \\ 1.07051016402631 \\ 2.85714285714286 \\ 0.0 \\ 0.0 \end{pmatrix}. \tag{1.41}$$

Solving this linear system gives us the required coefficients:

$$\begin{aligned} a_1 &= 2.55305620081729 & a_2 &= -2.76122280853939 \\ b_1 &= -7.02386028610914 & b_2 &= 14.6587935048229 \\ c_1 &= 6.38855196976641 & c_2 &= -23.0429533092186 \\ d_1 &= -1.91774788447456 & d_2 &= 11.3749950550874. \end{aligned} \tag{1.42}$$

The first cubic is for $x \in [1.0, 1.3]$, and the second cubic is for $x \in [1.3, 1.7]$.

For the active force length curve, we have three cubic segments: $f_1(x) : x \in [0.4, 0.7]$, $f_2(x) : x \in [0.7, 1.2]$, and $f_3(x) : x \in [1.2, 1.81]$. The conditions are:

$$\begin{aligned} &f_1(0.4) = 0.0, && f_1'(0.4) = 0.0, && f_2(1.0) = 1.0, \\ &f_2'(1.0) = 0.0, && f_3(1.81) = 0.0, && f_3'(1.81) = 0.0, \\ &f_1(0.7) - f_2(0.7) = 0, && f_1'(0.7) - f_2'(0.7) = 0, && f_2(1.2) - f_3(1.2) = 0, \\ &f_2'(1.2) - f_3'(1.2) = 0, && f_1(0.7) = 0.767020368396388, && f_2(1.2) = 0.89. \end{aligned} \tag{1.43}$$

The required coefficients are then:

$$\begin{aligned} a_1 &= -39.881247949014 & a_2 &= -0.322674853252855 & a_3 &= 4.85118460831817 \\ b_1 &= 68.3443204612586 & b_2 &= -1.71744046959073 & b_3 &= -20.9908796589089 \\ c_1 &= -35.5324573534802 & c_2 &= 4.40290549894003 & c_3 &= 28.3080866793169 \\ d_1 &= 5.83029153632759 & d_2 &= -1.36279017609645 & d_3 &= -11.2356843095252. \end{aligned} \tag{1.44}$$

The first cubic is for $x \in [0.4, 0.7]$, the second cubic is for $x \in [0.7, 1.2]$, and the third cubic is for $x \in [1.2, 1.81]$.

[4] These hard-coded values are extracted from OpenSim [Delp et al. 2007].

For the force velocity curve, we have three cubic segments: $f_1(x) : x \in [-1.0, -0.05]$, $f_2(x) : x \in [-0.05, 0.05]$, and $f_3(x) : x \in [0.05, 1.0]$. The conditions are:

$$\begin{aligned} &f_1(-1.0) = 0.0, && f_1'(-1.0) = 0.0, && f_1''(-1.0) = 0.0, \\ &f_3(1.0) = 1.4, && f_3'(1.0) = 0.0, && f_3''(1.0) = 0.0, \\ &f_1(-0.05) - f_2(-0.05) = 0, && f_1'(-0.05) - f_2'(-0.05) = 0, && f_2(0.05) - f_3(0.05) = 0, \\ &f_2'(0.05) - f_3'(0.05) = 0, && f_2(0.0) = 1.0, && f_2'(0.0) = 5.0. \end{aligned} \tag{1.45}$$

The required coefficients are then:

$$\begin{aligned} a_1 &= 0.915253473864379 & a_2 &= -453.333333333333 & a_3 &= 0.266648649866 \\ b_1 &= 2.74576042159314 & b_2 &= -8.78048780487805 & b_3 &= -0.799945949598001 \\ c_1 &= 2.74576042159314 & c_2 &= 5 & c_3 &= 0.799945949598001 \\ d_1 &= 0.915253473864378 & d_2 &= 1 & d_3 &= 1.133351350134. \end{aligned} \tag{1.46}$$

The first cubic is for $x \in [-1.0, -0.05]$, the second cubic is for $x \in [-0.05, 0.05]$, and the third cubic is for $x \in [0.05, 1.0]$.

To use these curves in Eq. (1.38), we need the normalized muscle length, $\tilde{l}$, and the normalized muscle velocity, $\tilde{v}$. These quantities are computed from the muscle path, which we assume is a straight line for now: given two points on two rigid bodies, the length is simply $l_{\text{MT}} = \|\Delta \boldsymbol{x}\|$. Out of this total musculotendon (MT) length, some portion of it is the tendon length, $l_{\text{T}}$, so the muscle length is $l = l_{\text{MT}} - l_{\text{T}}$. The normalized muscle length is obtained by dividing by the "optimal muscle length," which is often set to the rest length of the muscle:

$$\tilde{l} = \frac{l_{\text{MT}} - l_{\text{T}}}{l_{\text{opt}}}. \tag{1.47}$$

The muscle velocity, $v$, is computed as

$$v = \frac{\Delta \boldsymbol{x}^{\top}}{\|\Delta \boldsymbol{x}\|}(\dot{\boldsymbol{x}}_2 - \dot{\boldsymbol{x}}_1), \quad \dot{\boldsymbol{x}}_i = \mathsf{R}_i \Gamma_i \phi_i, \tag{1.48}$$

and the normalized muscle velocity is

$$\tilde{v} = \frac{v}{l_{\text{opt}} v_{\text{max}}}, \tag{1.49}$$

where $v_{\text{max}}$ is another parameter, the "maximum muscle contraction velocity." This gives us all the terms we need to compute the muscle force in Eq. (1.38).

Since the muscle force depends on both positions and velocities, we need to compute both its stiffness matrix, K, and its damping matrix, D. For the stiffness matrix, note first the similarity between Eq. (1.35) and Eq. (1.39). The difference between the muscle force and the zero rest-length force is that the muscle force has $\frac{f}{\|\Delta \boldsymbol{x}\|}$ in front of it, where as the zero rest-length force has just a constant $k$ in front of it. The stiffness matrix will thus have two terms:

$$\mathsf{K} = \underbrace{\frac{\partial}{\partial \mathsf{E}}\left(\frac{f}{\|\Delta \boldsymbol{x}\|}\right)\begin{pmatrix} \Gamma_1^{\top}\mathsf{R}_1^{\top}\Delta \boldsymbol{x} \\ -\Gamma_2^{\top}\mathsf{R}_2^{\top}\Delta \boldsymbol{x} \end{pmatrix}}_{\mathsf{K}_1} + \underbrace{\frac{f}{\|\Delta \boldsymbol{x}\|}\frac{\partial}{\partial \mathsf{E}}\begin{pmatrix} \Gamma_1^{\top}\mathsf{R}_1^{\top}\Delta \boldsymbol{x} \\ -\Gamma_2^{\top}\mathsf{R}_2^{\top}\Delta \boldsymbol{x} \end{pmatrix}}_{\mathsf{K}_2}. \tag{1.50}$$

The 2nd term has already been derived in Eq. (1.37):

$$\mathsf{K}_2 = \frac{f}{\|\Delta \boldsymbol{x}\|} \begin{pmatrix} [{}^1\boldsymbol{x}_1][\mathsf{R}_1^\top (\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & [{}^1\boldsymbol{x}_1] & [{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top \mathsf{R}_2[{}^2\boldsymbol{x}_2] & -[{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top \mathsf{R}_2 \\ [\mathsf{R}_1^\top (\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & I & \mathsf{R}_1^\top \mathsf{R}_2[{}^2\boldsymbol{x}_2] & -\mathsf{R}_1^\top \mathsf{R}_2 \\ [{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top \mathsf{R}_1[{}^1\boldsymbol{x}_1] & -[{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top \mathsf{R}_1 & [{}^2\boldsymbol{x}_2][\mathsf{R}_2^\top (\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & [{}^2\boldsymbol{x}_2] \\ \mathsf{R}_2^\top \mathsf{R}_1[{}^1\boldsymbol{x}_1] & -\mathsf{R}_2^\top \mathsf{R}_1 & [\mathsf{R}_2^\top (\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & I \end{pmatrix}. \tag{1.51}$$

The 1st term is the outer product between two vectors:

$$\mathsf{K}_1 = \begin{pmatrix} \Gamma_1^\top \mathsf{R}_1^\top \Delta \boldsymbol{x} \\ -\Gamma_2^\top \mathsf{R}_2^\top \Delta \boldsymbol{x} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_1 \Gamma_1 & -\boldsymbol{d}^\top \mathsf{R}_2 \Gamma_2 \end{pmatrix}, \tag{1.52}$$

where

$$\boldsymbol{d} = \frac{1}{l_{\mathrm{MT}}^2} \left( \frac{f}{l_{\mathrm{MT}}} - \frac{f_{\mathrm{opt}} \left( a f_{\mathrm{L}}' f_{\mathrm{V}} + f_{\mathrm{P}}' \right)}{l_{\mathrm{opt}}} \right) \Delta \boldsymbol{x}. \tag{1.53}$$

The damping matrix is the derivative of the force with respect to the velocity:

$$\mathsf{D} = \frac{\partial}{\partial \Phi} \left( \frac{f}{\|\Delta \boldsymbol{x}\|} \right) \begin{pmatrix} \Gamma_1^\top \mathsf{R}_1^\top \Delta \boldsymbol{x} \\ -\Gamma_2^\top \mathsf{R}_2^\top \Delta \boldsymbol{x} \end{pmatrix} \tag{1.54}$$

This will again be an outer product of two vectors:

$$\mathsf{D} = \begin{pmatrix} \Gamma_1^\top \mathsf{R}_1^\top \Delta \boldsymbol{x} \\ -\Gamma_2^\top \mathsf{R}_2^\top \Delta \boldsymbol{x} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_1 \Gamma_1 & -\boldsymbol{d}^\top \mathsf{R}_2 \Gamma_2, \end{pmatrix} \tag{1.55}$$

where

$$\boldsymbol{d} = -\frac{f_{\mathrm{opt}} a f_{\mathrm{L}} f_{\mathrm{V}}'}{l_{\mathrm{opt}} v_{\max} l_{\mathrm{MT}}^2} \Delta \boldsymbol{x}. \tag{1.56}$$

*1.9.5 Multi-Point Muscle.* Let the muscle path contain n points, each with its own rigid body. Then there are $n-1$ segments, from point $k$ to $k+1$. The scalar muscle force is computed using the total path of all the segments. Then this scalar force is used to multiply the normalized force within each segment.

The musculotendon length is now a summation, and the normalized muscle length is computed by subtracting the tendon length and dividing by the optimal muscle length as before:

$$l_{\mathrm{MT}} = \sum_{k=1}^{n-1} \|\Delta \boldsymbol{x}_k\|, \quad \tilde{l} = \frac{l_{\mathrm{MT}} - l_{\mathrm{T}}}{l_{\mathrm{opt}}}, \tag{1.57}$$

where $\Delta \boldsymbol{x}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$. The muscle velocity is also a summation:

$$v = \sum_{k=1}^{n-1} \frac{\Delta \boldsymbol{x}_k^\top}{\|\Delta \boldsymbol{x}_k\|} \Delta \dot{\boldsymbol{x}}_k, \quad \tilde{v} = \frac{v}{l_{\mathrm{opt}} v_{\max}}, \tag{1.58}$$

where $\Delta \dot{\boldsymbol{x}}_k = \dot{\boldsymbol{x}}_{k+1} - \dot{\boldsymbol{x}}_k$. The scalar muscle force is then computed with Eq. (1.38):

$$f = f_{\mathrm{opt}} \left( a f_{\mathrm{L}}(\tilde{l}) f_{\mathrm{V}}(\tilde{v}) + f_{\mathrm{P}}(\tilde{l}) \right). \tag{1.59}$$

The generalized forces are then:

$$\mathsf{f} = f \sum_{k=1}^{n-1} \mathsf{f}_k, \quad \mathsf{f}_k = \frac{1}{\|\Delta \boldsymbol{x}_k\|} \begin{pmatrix} \Gamma_k^\top \mathsf{R}_k^\top \Delta \boldsymbol{x}_k \\ -\Gamma_{k+1}^\top \mathsf{R}_{k+1}^\top \Delta \boldsymbol{x}_k \end{pmatrix}. \tag{1.60}$$

The scalar muscle force, $f$, is the same for all the segments, and the portion inside the sum is the normalized force within each segment.

The stiffness matrix can be computed as

$$\mathsf{K} = \frac{\partial f}{\partial \mathsf{E}} \sum_{k=1}^{n-1} \mathsf{f}_k + f \sum_{k=1}^{n-1} \mathsf{K}_k \tag{1.61}$$

As before, we can compute the stiffness matrix for the normalized force in two parts:

$$\begin{aligned}
&\mathsf{K}_k = \mathsf{K}_{k1} + \mathsf{K}_{k2} \\
&\mathsf{K}_{k1} = \begin{pmatrix} \Gamma_k^\top \mathsf{R}_k^\top \Delta\boldsymbol{x} \\ -\Gamma_{k+1}^\top \mathsf{R}_{k+1}^\top \Delta\boldsymbol{x} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_k \Gamma_k & -\boldsymbol{d}^\top \mathsf{R}_{k+1} \Gamma_{k+1} \end{pmatrix}, \quad \boldsymbol{d} = \frac{\Delta\boldsymbol{x}_k}{\|\Delta\boldsymbol{x}_k\|^3} \\
&\mathsf{K}_{k2} = \frac{1}{\|\Delta\boldsymbol{x}_k\|} \begin{pmatrix} [{}^k\boldsymbol{x}_k][\mathsf{R}_k^\top (\boldsymbol{p}_k - {}^0\boldsymbol{x}_{k+1})] & [{}^k\boldsymbol{x}_k] & [{}^k\boldsymbol{x}_k]\mathsf{R}_k^\top \mathsf{R}_{k+1}[{}^{k+1}\boldsymbol{x}_{k+1}] & -[{}^k\boldsymbol{x}_k]\mathsf{R}_k^\top \mathsf{R}_{k+1} \\ [\mathsf{R}_k^\top (\boldsymbol{p}_k - {}^0\boldsymbol{x}_{k+1})] & I & \mathsf{R}_k^\top \mathsf{R}_{k+1}[{}^{k+1}\boldsymbol{x}_{k+1}] & -\mathsf{R}_k^\top \mathsf{R}_{k+1} \\ [{}^{k+1}\boldsymbol{x}_{k+1}]\mathsf{R}_{k+1}^\top \mathsf{R}_k[{}^k\boldsymbol{x}_k] & -[{}^{k+1}\boldsymbol{x}_{k+1}]\mathsf{R}_{k+1}^\top \mathsf{R}_k & [{}^{k+1}\boldsymbol{x}_{k+1}][\mathsf{R}_{k+1}^\top (\boldsymbol{p}_{k+1} - {}^0\boldsymbol{x}_k)] & [{}^{k+1}\boldsymbol{x}_{k+1}] \\ \mathsf{R}_{k+1}^\top \mathsf{R}_k[{}^k\boldsymbol{x}_k] & -\mathsf{R}_{k+1}^\top \mathsf{R}_k & [\mathsf{R}_{k+1}^\top (\boldsymbol{p}_{k+1} - {}^0\boldsymbol{x}_k)] & I \end{pmatrix}.
\end{aligned} \tag{1.62}$$

The first term of Eq. (1.61) is an outer product between two vectors of length $6n$, since the summation, when expanded, will form a single long vector, with each summand contributing to two block locations. The left expression, $\frac{\partial f}{\partial \mathsf{E}}$, is also evaluated by summing over the segments:

$$\frac{\partial f}{\partial \mathsf{E}} = \sum_{k=1}^{n-1} \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_k \Gamma_k & -\boldsymbol{d}^\top \mathsf{R}_{k+1} \Gamma_{k+1} \end{pmatrix}, \quad \boldsymbol{d} = \frac{-f_{\text{opt}}(a f_{\text{L}}' f_{\text{V}} + f_{\text{P}}')}{l_{\text{opt}}} \frac{\Delta\boldsymbol{x}_k}{\|\Delta\boldsymbol{x}_k\|}. \tag{1.63}$$

The damping matrix also is an outer product between two vectors of length $6n$, constructed by iterating over the segments.

$$\begin{aligned}
&\mathsf{D} = \frac{\partial f}{\partial \Phi} \sum_{k=1}^{n-1} \mathsf{f}_k \\
&\frac{\partial f}{\partial \Phi} = \sum_{k=1}^{n-1} \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_k \Gamma_k & -\boldsymbol{d}^\top \mathsf{R}_{k+1} \Gamma_{k+1} \end{pmatrix}, \quad \boldsymbol{d} = \frac{-f_{\text{opt}}(a f_{\text{L}} f_{\text{V}}')}{l_{\text{opt}} v_{\max}} \frac{\Delta\boldsymbol{x}_k}{\|\Delta\boldsymbol{x}_k\|}.
\end{aligned} \tag{1.64}$$

*1.9.6 Spring Damper.* Spring-damper is a force between two points that tries to maintain its rest length, $L$:

$$\boldsymbol{f} = \left( k \frac{l - L}{L} - d \frac{\dot{l}}{L} \right) \frac{\Delta\boldsymbol{x}}{l}, \quad l = \|\Delta\boldsymbol{x}\|, \tag{1.65}$$

for stiffness parameter, $k$, and damping parameter, $d$. The time derivative of length, $\dot{l}$ is given in Eq. (1.48) (same as $v$). As before, we will separate this into scalar and vector components:

$$\mathsf{f} = f_s \mathsf{f}_n, \quad f_s = \left( k \frac{l - L}{L} - d \frac{\dot{l}}{L} \right), \quad \mathsf{f}_n = \frac{1}{l} \begin{pmatrix} -\Gamma_1^\top \mathsf{R}_1^\top \Delta\boldsymbol{x} \\ \Gamma_2^\top \mathsf{R}_2^\top \Delta\boldsymbol{x} \end{pmatrix}. \tag{1.66}$$

The stiffness matrix is then

$$\begin{aligned}
\mathsf{K} &= \frac{\partial f_s}{\partial \mathsf{E}} \mathsf{f}_n + f_s \left(\mathsf{K}_{n1} + \mathsf{K}_{n2}\right) \\
\mathsf{K}_{n1} &= \begin{pmatrix} \Gamma_1^\top \mathsf{R}_1^\top \Delta \boldsymbol{x} \\ -\Gamma_2^\top \mathsf{R}_2^\top \Delta \boldsymbol{x} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_1 \Gamma_1 & -\boldsymbol{d}^\top \mathsf{R}_2 \Gamma_2 \end{pmatrix}, \quad \boldsymbol{d} = \frac{\Delta \boldsymbol{x}}{l^3} \\
\mathsf{K}_{n2} &= \frac{1}{l} \begin{pmatrix} [{}^1\boldsymbol{x}_1][\mathsf{R}_1^\top (\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & [{}^1\boldsymbol{x}_1] & [{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top \mathsf{R}_2 [{}^2\boldsymbol{x}_2] & -[{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top \mathsf{R}_2 \\ [\mathsf{R}_1^\top (\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & I & \mathsf{R}_1^\top \mathsf{R}_2 [{}^2\boldsymbol{x}_2] & -\mathsf{R}_1^\top \mathsf{R}_2 \\ [{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top \mathsf{R}_1 [{}^1\boldsymbol{x}_1] & -[{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top \mathsf{R}_1 & [{}^2\boldsymbol{x}_2][\mathsf{R}_2^\top (\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & [{}^2\boldsymbol{x}_2] \\ \mathsf{R}_2^\top \mathsf{R}_1 [{}^1\boldsymbol{x}_1] & -\mathsf{R}_2^\top \mathsf{R}_1 & [\mathsf{R}_2^\top (\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & I \end{pmatrix} \\
\frac{\partial f_s}{\partial \mathsf{E}} &= \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_1 \Gamma_1 & -\boldsymbol{d}^\top \mathsf{R}_2 \Gamma_2 \end{pmatrix}, \quad \boldsymbol{d} = \frac{-k}{L} \frac{\Delta \boldsymbol{x}}{l}.
\end{aligned} \tag{1.67}$$

The damping matrix is

$$\mathsf{D} = \frac{\partial f_s}{\partial \Phi} \mathsf{f}_n, \quad \frac{\partial f_s}{\partial \Phi} = \begin{pmatrix} \boldsymbol{d}^\top \mathsf{R}_1 \Gamma_1 & -\boldsymbol{d}^\top \mathsf{R}_2 \Gamma_2 \end{pmatrix}, \quad \boldsymbol{d} = \frac{-d}{L} \frac{\Delta \boldsymbol{x}}{l}. \tag{1.68}$$

## 1.10 Joint Constraints

Joint constraints between rigid bodies are implemented using the adjoint formulation [Murray et al. 1994], from which we can easily derive various types of joints simply by dropping different rows in the $6 \times 6$ adjoint matrix. Given two rigid bodies, $i$ and $k$, and a joint frame defined with respect to the first body, ${}^i_j\mathsf{E}$, we constrain the rigid bodies' spatial velocities, $\phi_i$ and $\phi_k$, with respect to the joint frame. Using Eq. (1.15), the relative velocity at joint $j$ is given by

$$\begin{aligned}
\delta\phi_j &= {}^j_i\mathrm{Ad}\, \phi_i - {}^j_k\mathrm{Ad}\, \phi_k \\
&= \begin{pmatrix} {}^j_i\mathrm{Ad} & -{}^j_i\mathrm{Ad}\, {}^i_0\mathrm{Ad}\, {}^0_k\mathrm{Ad} \end{pmatrix} \begin{pmatrix} \phi_i \\ \phi_k \end{pmatrix}.
\end{aligned} \tag{1.69}$$

(Note ${}^i_0\mathrm{Ad} = {}^0_i\mathrm{Ad}^{-1}$.) For a rigid joint, we want the relative velocities to be zero, so we set $\delta\phi = 0$. From this, we can derive different types of joints, by dropping various rows of the constraint equation: for example, the top three rows (corresponding to the three rotational DoFs) for a ball joint, or the third row (corresponding to the rotation about the z-axis) for a hinge joint.

Setting $\delta\phi = 0$, we can write Eq. (1.69) in matrix form as $\mathsf{G}\Phi = 0$, where $\mathsf{G} \in \mathbb{R}^{6\times 12}$ and $\Phi \in \mathbb{R}^{12\times 1}$. (This is for a fixed joint. For a hinge joint, $\mathsf{G} \in \mathbb{R}^{5\times 12}$.) As we add more bodies and joints, we add more rows to this "constraint" matrix. For example, if we have 3 bodies and 2 hinge joints between them, then G will have $5 + 5 = 10$ rows and $6 + 6 + 6 = 18$ columns. The entries in G need to line up, so that the correct terms get multiplied with each other. For example, if the 2 hinge joints are between bodies 1 and 2, and between 1 and 3, the constraint equation is

$$\begin{pmatrix} \mathsf{G}_{11} & \mathsf{G}_{12} & 0 \\ \mathsf{G}_{21} & 0 & \mathsf{G}_{23} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{1.70}$$

Before we incorporate these constraints into dynamics, let's first simplify the notation of Eq. (1.28), by letting $\phi_i = \phi_i^{(k+1)}$ and $\tilde{\mathsf{f}}_i = \mathsf{M}_i \phi_i^{(k)} + h\left(\mathrm{ad}(\phi_i^{(k)})^\top \mathsf{M}_i \phi_i^{(k)} + \mathsf{f}_{\text{body}}\left(\mathsf{E}_i^{(k)}\right)\right)$. Then we have $\mathsf{M}_i \phi_i = \tilde{\mathsf{f}}_i$, which is a $6 \times 6$ linear system. If we combine all bodies, we get $\mathsf{M}\Phi = \tilde{\mathsf{f}}$, which is a $6n \times 6n$ linear system. We can think of this linear system as

the solution to the following quadratic minimization problem:

$$\underset{\Phi}{\text{minimize}} \quad \frac{1}{2}\Phi^\top \mathsf{M}\Phi - \Phi^\top \tilde{\mathsf{f}}. \tag{1.71}$$

To this quadratic objective, we add the equality constraint equation $\mathsf{G}\Phi = 0$, giving us

$$\begin{aligned} \underset{\Phi}{\text{minimize}} \quad & \frac{1}{2}\Phi^\top \mathsf{M}\Phi - \Phi^\top \tilde{\mathsf{f}} \\ \text{subject to} \quad & \mathsf{G}\Phi = 0. \end{aligned} \tag{1.72}$$

Since the mass matrix, $\mathsf{M}$, is always positive (semi) definite, the objective is convex, and using duality, we can convert this quadratic minimization problem into a single linear system, called a Karush-Kuhn-Tucker (KKT) system [Boyd and Vandenberghe 2004].

$$\begin{pmatrix} \mathsf{M} & \mathsf{G}^\top \\ \mathsf{G} & 0 \end{pmatrix} \begin{pmatrix} \Phi \\ \lambda \end{pmatrix} = \begin{pmatrix} \tilde{\mathsf{f}} \\ 0 \end{pmatrix}. \tag{1.73}$$

The top entry in the solution vector, $\Phi$, is the new velocities of all the rigid bodies, and the bottom entry in the solution vector, $\lambda$, is the vector of Lagrange multipliers for the constraints. The joint reaction forces can be computed as $-\mathsf{G}^\top\lambda/h$. For intuition, the top and bottom rows can be rewritten separately:

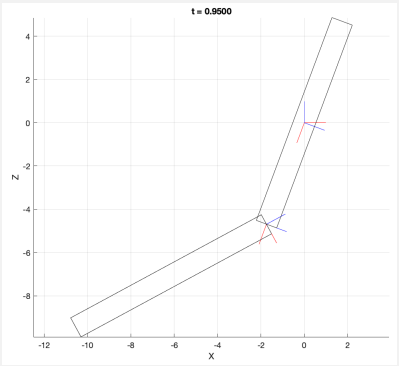
$$\begin{aligned} \mathsf{M}\Phi + \mathsf{G}^\top\lambda &= \tilde{\mathsf{f}} \\ \mathsf{G}\Phi &= 0. \end{aligned} \tag{1.74}$$

The top equation is the original discretized Newton-Euler equation but with constraint forces added, and the bottom equation is the constraint equation from the joints.

Run the sample simulation code, `testJoint.m`:

```
>> testJoint
rigid1: 10.00 g
rigid2: 10.00 g
```

The first body has a joint wrt to the world, and the second body has a joint wrt to the first body.




## 1.11 Contact Constraints

First, let's assume that a single body is colliding with the ground. A collision detector, which is outside the scope of this document, returns the collision point and the collision normal. Let's assume that these are both in world coordinates: ${}^0\boldsymbol{x}$ and ${}^0\boldsymbol{n}$. What we require from the colliding rigid body is that the velocity of the collision point be positive wrt the collision normal. Using the material Jacobian, Eq. (1.12), the velocity of the collision point is

$${}^0\dot{\boldsymbol{x}} = {}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x})\,{}^i\phi_i, \tag{1.75}$$

where ${}^i\boldsymbol{x} = {}^i_0\mathsf{E}\,{}^0\boldsymbol{x}$ is the collision point in body local coordinates. We want this world velocity to be positive wrt to the collision normal, so the final constraint is

$$\begin{aligned} {}^0\boldsymbol{n}^\top\, {}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x})\,{}^i\phi_i \geq 0 \qquad \text{or} \\ \mathsf{N}\,{}^i\phi_i \geq 0, \end{aligned} \tag{1.76}$$

where $\mathsf{N} = \boldsymbol{n}^\top\,\mathsf{R}\,\Gamma$. In this simple case of a single collision point on a single body, $\mathsf{N}$ is a $1 \times 6$ matrix. If we have multiple contact points, we can add more rows to this constraint matrix, with each row having a slightly different entry because the contact point, ${}^i\boldsymbol{x}$, is going to be different. If the collision occurs with other objects in the scene (*i.e.*, not the ground), the collision normal may also be different. If there are multiple rigid bodies colliding with the world, then $\mathsf{N}$ will be of size $m \times 6n$, where $m$ is the total number of collisions, and $n$ is the number of rigid bodies.

Now let's see how we handle collisions between bodies $i$ and $j$. What we do now is to constrain the *relative* velocity between the colliding bodies. The collision detector (usually) doesn't know whether things are moving, so it will just return a list of collision points and normals as before.[5] As before, we have collision point ${}^0\boldsymbol{x}$ and normal ${}^0\boldsymbol{n}$. The relative velocity, ${}^0\boldsymbol{v}_{\text{rel}}$, between the two points in contact, expressed in world coordinates is

$${}^0\boldsymbol{v}_{\text{rel}} = {}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x})\,{}^i\phi_i - {}^0_j\mathsf{R}\,\Gamma({}^j\boldsymbol{x})\,{}^j\phi_j, \tag{1.77}$$

where ${}^i\boldsymbol{x} = {}^i_0\mathsf{E}\,{}^0\boldsymbol{x}$ and ${}^j\boldsymbol{x} = {}^j_0\mathsf{E}\,{}^0\boldsymbol{x}$. We want this relative velocity to be positive wrt the collision normal: ${}^0\boldsymbol{n}^\top\,{}^0\boldsymbol{v}_{\text{rel}} \geq 0$. In matrix form, we have the following:

$$\begin{pmatrix} {}^0\boldsymbol{n}^\top\,{}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x}) & -{}^0\boldsymbol{n}^\top\,{}^0_j\mathsf{R}\,\Gamma({}^j\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} {}^i\phi_i \\ {}^j\phi_j \end{pmatrix} \geq 0. \tag{1.78}$$

Each collision between bodies takes 2 block columns (12 columns total) of the $\mathsf{C}$ matrix. By combining all collisions into the contact constraint matrix, we can write $\mathsf{C}\Phi \geq 0$. For an example filling pattern, see Eq. (1.70).

By adding the constraint to Eq. (1.71), we obtain the following quadratic program:

$$\begin{aligned} \underset{\Phi}{\text{minimize}} \quad & \frac{1}{2}\Phi^\top\mathsf{M}\Phi - \Phi^\top\tilde{\mathsf{f}} \\ \text{subject to} \quad & \mathsf{C}\Phi \geq 0. \end{aligned} \tag{1.79}$$

If there are joint constraints as well, we must solve

$$\begin{aligned} \underset{\Phi}{\text{minimize}} \quad & \frac{1}{2}\Phi^\top\mathsf{M}\Phi - \Phi^\top\tilde{\mathsf{f}} \\ \text{subject to} \quad & \mathsf{C}\Phi \geq 0 \\ & \mathsf{G}\Phi = 0. \end{aligned} \tag{1.80}$$

## 1.12 Stabilization

Since the constraints are applied at the acceleration or the velocity level, there is an unavoidable constraint drift. Even though the joints are initially intact, over time, they will come apart because no position information is used during the integration. There are two basic methods for combating this drift problem: Baumgarte stabilization [Baumgarte 1972] and post stabilization [Cline and Pai 2003]. We'll look at Baumgarte stabilization because it is general, simple, and computationally efficient. (Post-stabilization works better in some cases though.)

[5] *Continuous* collision detector also takes into account velocities.

To start, we have a positional constraint, $g(\mathsf{q}) = 0$, where $\mathsf{q}$ is the vector of positional (or configuration) DOFs. With maximal coordinates, there are many parameter choices for rigid body configuration, but we choose to use $\mathsf{E}$, the 4x4 transformation matrix, so the positional constraint can also be written as $g(\mathsf{E}) = 0$. If we differentiate this wrt time, we get the velocity-level constraint equation that we have been using so far:

$$\begin{aligned}\frac{dg}{dt} &= \frac{\partial g}{\partial \mathsf{q}}\dot{\mathsf{q}} \\ &= \mathsf{G}\Phi,\end{aligned} \tag{1.81}$$

which we set to 0.

Baumgarte stabilization works by adding back the positional information into the velocity- or acceleration-level constraint. Baumgarte's original formulation is

$$\begin{aligned}\mathsf{G}\dot{\mathsf{q}} &= -\gamma g \\ \mathsf{G}\ddot{\mathsf{q}} &= -\dot{\mathsf{G}}\dot{\mathsf{q}} - 2\alpha\mathsf{G}\dot{\mathsf{q}} - \beta^2 g,\end{aligned} \tag{1.82}$$

respectively for velocity- and acceleration-level stabilization. In terms of implementation, this means that the RHS constraint vector in the KKT system or the quadratic program get replaced by the RHS values in the equations above. The values for $\alpha$, $\beta$, and $\gamma$ must be chosen experimentally. For example, Baumgarte uses $\alpha = \beta = \gamma = 10$ in some of his experiments. Sometimes, to make the units work out, $\alpha = \beta = \gamma = 1/h$ is used.

### 1.13 Friction

Here, we cover the single-QP friction model, introduced by Anitescu and Hart [2004]. This QP has an equivalent LCP (linear complementarity problem) formulation, which is often solved using an iterative Gauss-Seidel approach. For more accurate friction, we need to solve a more difficult mathematical problem. See Staggered Projections [Kaufman et al. 2008] for an example.

We start with the contact only (no friction) QP formulation from §1.11. In Equation Eq. (1.79), the "primal" variables are the rigid body velocities, and the "dual" variables are the Lagrange multipliers for the contact constraints [Boyd and Vandenberghe 2004]. We can form the equivalent, dual version of this QP as follows:

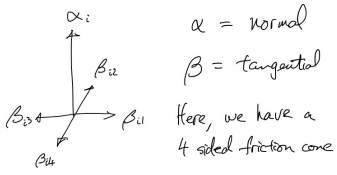
$$\begin{aligned}\underset{\alpha}{\text{minimize}} \quad & \frac{1}{2}\alpha^\top \mathsf{N}\mathsf{M}^{-1}\mathsf{N}^\top\alpha + \alpha^\top \mathsf{N}\mathsf{M}^{-1}\tilde{\mathsf{f}} \\ \text{subject to} \quad & \alpha \geq 0.\end{aligned} \tag{1.83}$$

Once we compute $\alpha$, we can solve for $\Phi$ with:

$$\mathsf{M}\Phi + \mathsf{N}^\top\alpha = \tilde{\mathsf{f}}. \tag{1.84}$$

The dual variables, $\alpha$, are the contact impulse magnitudes. Since contact constraints can only push, $\alpha$ must be positive. The product $\mathsf{N}^\top\alpha$ represents the generalized force (wrench) that the contact constraints apply to the rigid bodies.

To enable friction, we need to introduce the tangential impulse, $\beta$. Just like how $\alpha$ can only apply contact impulses in the normal direction, $\beta$ can only apply contact impulses in the tangential direction. See the inset figure for an example. Here, we are using a four-sided pyramid the approximate the "friction cone." In this case, for each contact constraint, there are four tangential constraints. This means that the "tangent" matrix, $\mathsf{T}$, is going to have four times as many rows as the normal matrix, $\mathsf{N}$. Just like the normal impulses, the tangential impulses must be



positive. On top of this, we need an additional constraint to ensure that the tangential impulse magnitude is less than the normal impulse magnitude multiplied by the coefficient of friction: $f_{\parallel} \leq \mu f_{\perp}$. With the four-sided pyramid, this can be expressed as:

$$\begin{pmatrix} \mu & -1 & -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{pmatrix} \geq 0. \tag{1.85}$$

This linear relationship between $\alpha$ and $\beta$ can be expressed with a matrix $E$. For example, if there are two contact points, $E$ becomes:

$$\begin{pmatrix} \mu & 0 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \beta_{11} \\ \beta_{12} \\ \beta_{13} \\ \beta_{14} \\ \beta_{21} \\ \beta_{22} \\ \beta_{23} \\ \beta_{24} \end{pmatrix} \geq 0. \tag{1.86}$$

Let's combine the normal and tangential dual variables into one:

$$\lambda = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \quad \mathsf{C} = \begin{pmatrix} \mathsf{N} \\ \mathsf{T} \end{pmatrix}. \tag{1.87}$$

Then, the final single-QP friction problem is

$$\begin{aligned} \underset{\lambda}{\text{minimize}} \quad & \frac{1}{2}\lambda^{\top}\mathsf{C}\mathsf{M}^{-1}\mathsf{C}^{\top}\lambda + \lambda^{\top}\mathsf{C}\mathsf{M}^{-1}\tilde{\mathsf{f}} \\ \text{subject to} \quad & \lambda \geq 0, \\ & E\lambda \geq 0. \end{aligned} \tag{1.88}$$



## 2 REDUCED COORDINATES

Forward dynamics with reduced coordinates was originally developed for robotics applications [Featherstone 1983; Park et al. 1995]. Unlike the maximal coordinate formulation, which requires $6n$ degrees of freedom (DOFs) and $5n$ constraints (for revolute/hinge joints), the reduced coordinate formulation requires only $n$ DOFs and no constraints. Initially, linear time algorithm was known only for reduced coordinates, but Baraff [1996] proposed a linear time algorithm that uses maximal coordinates. Baraff argues that one of the advantages of maximal coordinates is that they're easier to understand and implement.

> "While $O(n)$ *inverse* reduced-coordinate approaches are easily understood, *forward* reduced-coordinate formulations with linear time complexity have an extremely steep learning curve, and make use of a formidable array of notational tools. The author admits (as do many practitioners the author has queried) to lacking a solid, intuitive understanding of these methods."

The reason inverse dynamics is easier than forward dynamics can be seen by looking at $\mathsf{f} = \mathsf{M}\ddot{\mathsf{q}}$. In inverse dynamics, we are given $\ddot{\mathsf{q}}$ and solve for $\mathsf{f}$, whereas in forward dynamics, we are given $\mathsf{f}$ and solve for $\ddot{\mathsf{q}}$. Therefore, with inverse dynamics, we need to know how to multiply by the mass matrix, $\mathsf{M}$, whereas with forward dynamics, we need to know how to multiply by the *inverse* mass matrix, $\mathsf{M}^{-1}$, or *solve* with the mass matrix.

Baraff goes on to argue for the use of maximal coordinates, since the resulting KKT matrix (Eq. (1.73)) can be factored in linear time as long as there are no loops. (Loops can be supported for a small cost.) Baraff does mention an important advantage of using reduced coordinates—lack of constraint drift. However, combining reduced coordinates with other types of simulation (*e.g.,* FEM) is again challenging. What we present here is not linear time, but is easy to understand and implement.

The main thing we need is a way to map between reduced coordinates and maximal coordinates. For now, let's assume that we only have revolute (hinge) joints, so our reduced coordinates are composed of a series of joint angles. We'll also assume that there are no loops in the mechanism. Let $\mathsf{q}_r$, $\dot{\mathsf{q}}_r$, and $\ddot{\mathsf{q}}_r$ be the reduced position, velocity, and acceleration. As before, for maximal coordinates, we'll also use $\mathsf{E}$, $\phi$, and $\dot{\phi}$ for individual rigid bodies, but we'll use $\mathsf{q}_m$, $\dot{\mathsf{q}}_m$, $\ddot{\mathsf{q}}_m$, for the stacked vectors of all rigid bodies. Our goal here is to find a Jacobian $\mathsf{J}_{mr}$ that maps generalized coordinates to maximal coordinates:

$$\dot{\mathsf{q}}_m = \mathsf{J}_{mr}\dot{\mathsf{q}}_r. \tag{2.1}$$

Once we derive this Jacobian, we will be ready to start working out the dynamics in reduced coordinates. We start with the Newton-Euler equations of motion of rigid bodies in maximal coordinates Eq. (1.21). Instead of a single body, assume we have a system of bodies in the matrix form $\mathsf{M}_m\ddot{\mathsf{q}}_m = \mathsf{f}_m$, where $\mathsf{f}_m$ contains all forces including Coriolis forces. This system has $6n$ degrees of freedom, since it is in maximal coordinates. Using the Jacobian in Eq. (2.14), we can convert this into reduced coordinates. First, we need the mapping between reduced and maximal accelerations:

$$\ddot{\mathsf{q}}_m = \dot{\mathsf{J}}_{mr}\dot{\mathsf{q}}_r + \mathsf{J}_{mr}\ddot{\mathsf{q}}_r. \tag{2.2}$$

Therefore, we need not only the Jacobian, $\mathsf{J}$, but also its time derivative, $\dot{\mathsf{J}}$.

The Jacobian is easier to understand in terms of velocities, but we'll start with positions. We'll first assume that the joint hierarchy forms an acyclic graph, *i.e.,* a tree. If the system has loops, we first need to break them so that we get a spanning tree, and we will put these loops back later with constraints in §2.8. In a tree, each node only has one parent, with the root node having a null parent. So, we can assume that there is a one-to-one mapping between a body and a joint, and therefore we can pretty much use "body" and "joint" interchangeably. Joints and bodies always come in pairs, and for each pair, the body is always understood to be the child body connected to the joint.

For now, we'll take a joint-centric view. For any joint $j$, its transformation matrix, or configuration in world, ${}^0_j\mathsf{E}$, can be computed by chaining the transforms matrices from the root to the joint. For a serial chain, we get

$${}^0_j\mathsf{E} = {}^0_1\mathsf{E}\,{}^1_2\mathsf{E}\cdots{}^{j-1}_j\mathsf{E}. \tag{2.3}$$

Each joint transform is a function of $\mathsf{q}$. For example, for a revolute joint about the Z-axis, we have

$${}^{j-1}_j\mathsf{E} = {}^{j-1}_j\mathsf{E}_0\;\mathsf{Q}_j(\mathsf{q}_j), \quad \mathsf{Q}_j(\mathsf{q}_j) = \begin{pmatrix} \cos(\mathsf{q}_j) & -\sin(\mathsf{q}_j) & 0 & 0 \\ \sin(\mathsf{q}_j) & \cos(\mathsf{q}_j) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.4}$$

where ${}^{j-1}_{j}\mathsf{E}_0$ is a translation matrix that represents where $j$ is wrt $j-1$, which does not change at run time. The rotation matrix, $\mathrm{Q}_j(\mathrm{q}_j)$, then applies the actual transformation as a function of $\mathrm{q}_j$. When we derive the Jacobian, rather than using a rotation matrix directly as above, we will be using the matrix exponential, since we'll be working with spatial velocities. The rotation matrix above can be written equivalently as

$$\mathrm{Q}_j(\mathrm{q}_j) = \exp\left(\left[S\mathrm{q}_j\right]\right), \quad S = \begin{pmatrix}0 & 0 & 1 & 0 & 0 & 0\end{pmatrix}^\top. \tag{2.5}$$

In other words, the product $S\mathrm{q}_j$ gets constructed into a skew symmetric matrix using Eq. (1.6), which is then exponentiated to construct a transformation matrix.

To get the transform of the body attached to the joint, we right multiply by an additional transform, ${}^{j}_{B_j}\mathsf{E}$, that represents where the body is wrt the joint, which again does not change at run time.

$${}^{0}_{B}\mathsf{E} = {}^{0}_{1}\mathsf{E}\,{}^{1}_{2}\mathsf{E}\cdots{}^{j-1}_{j}\mathsf{E}\,{}^{j}_{B_j}\mathsf{E}, \tag{2.6}$$

where $B_j$ is the body attached to the joint $j$.

## 2.1 Jacobian

Let's now derive the Jacobian in terms of maximal and reduced velocities: $\Phi = \mathrm{J}\dot{\mathrm{q}}$. Unlike the previous section, we'll be taking a body-centric view. Say we have a body $i$ and its parent $p$, and there is a joint $J_i$ between them. As we saw in §1.10, we can compute the relative twist between $p$ and $i$ at $J_i$. The twist of $p$ and $i$ at $J_i$ are

$${}^{J_i}\phi_p = {}^{J_i}_{p}\mathrm{Ad}\,{}^{p}\phi_p, \quad {}^{J_i}\phi_i = {}^{J_i}_{i}\mathrm{Ad}\,{}^{i}\phi_i, \tag{2.7}$$

and their relative twist is

$$\begin{aligned}{}^{J_i}\phi_{J_i} &= {}^{J_i}\phi_i - {}^{J_i}\phi_p \\ &= {}^{J_i}_{i}\mathrm{Ad}\,{}^{i}\phi_i - {}^{J_i}_{p}\mathrm{Ad}\,{}^{p}\phi_p \\ &= {}^{J_i}_{i}\mathrm{Ad}\,{}^{i}\phi_i - {}^{J_i}_{i}\mathrm{Ad}\,{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}\,{}^{p}\phi_p,\end{aligned} \tag{2.8}$$

where 0 indicates the world frame. Since $i$ owns the joint, ${}^{J_i}_{i}\mathrm{Ad}$ is constant. (It's constructed from ${}^{J_i}_{i}\mathsf{E}$, which represents where the $i$'s body center is wrt to the joint, which is set at initialization.) Remember that in maximal coordinates, we store positions wrt the world and velocities wrt the body itself. In other words, for each body, we store ${}^{0}_{i}\mathsf{E}$ and ${}^{i}\phi_i$. So in the above expression, the adjoint matrices of the form ${}^{0}_{i}\mathrm{Ad}$ and ${}^{i}_{0}\mathrm{Ad}$ can be computed easily from ${}^{0}_{i}\mathsf{E}$. We can rearrange this to solve for $i$'s spatial velocity.

$$\begin{aligned}{}^{J_i}_{i}\mathrm{Ad}\,{}^{i}\phi_i &= {}^{J_i}_{i}\mathrm{Ad}\,{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}\,{}^{p}\phi_p + {}^{J_i}\phi_{J_i} \\ {}^{i}\phi_i &= {}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}\,{}^{p}\phi_p + {}^{i}_{J_i}\mathrm{Ad}\,{}^{J_i}\phi_j.\end{aligned} \tag{2.9}$$

What this expression implies is that if we know the parent's velocity, ${}^{p}\phi_p$, and the joint's velocity, ${}^{j}\phi_j$, we can compute the child's velocity, ${}^{i}\phi_i$. In reduced coordinates, we parameterize ${}^{j}\phi_j$ not with the full 6 degrees of freedom but with some subset $\subset \mathbb{R}^6$. For example, assuming we're using revolute joints about the Z axis, we can write

$${}^{J_i}\phi_{J_i} = S\dot{\mathrm{q}}_i, \quad S = \begin{pmatrix}0 & 0 & 1 & 0 & 0 & 0\end{pmatrix}^\top. \tag{2.10}$$

We use $S$ here to follow the notation of Park et al. [1995] and Kim [2012]. $S$ takes on this simple form for revolute joints, but it gets quite complicated for spherical joints, as we'll see later in §2.4. Combining Eq. (2.9) and Eq. (2.10), we get

$$ {}^i\phi_i = \underbrace{{}^i_0\mathrm{Ad}\,{}^0_p\mathrm{Ad}}_{{}^i_p\mathrm{Ad}}\,{}^p\phi_p + \underbrace{{}^i_{J_i}\mathrm{Ad}\,S}_{{}^i_{J_i}\mathrm{Ad}_S}\dot{\mathrm{q}}_i \tag{2.11} $$

This relationship can be recursively applied to get the system Jacobian. Let's simplify the notation a bit by defining the two factors above as ${}^i_p\mathrm{Ad}$ and ${}^i_{J_i}\mathrm{Ad}_S$. Let's also use 0, 1, 2, ..., instead of $p$ and $i$. So we have

$$ {}^1\phi_1 = {}^1_0\mathrm{Ad}\,{}^0\phi_0 + {}^1_{J_1}\mathrm{Ad}_S\,\dot{\mathrm{q}}_1, \tag{2.12} $$

but we can assume that the world frame is stationary, so ${}^0\phi_0 = 0$. Continuing recursively,

$$ \begin{aligned} {}^2\phi_2 &= {}^2_1\mathrm{Ad}\,{}^1\phi_1 + {}^2_{J_2}\mathrm{Ad}_S\,\dot{\mathrm{q}}_2 \\ &= {}^2_1\mathrm{Ad}\left({}^1_{J_1}\mathrm{Ad}_S\,\dot{\mathrm{q}}_1\right) + {}^2_{J_2}\mathrm{Ad}_S\,\dot{\mathrm{q}}_2 \\ &= {}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S\,\dot{\mathrm{q}}_1 + {}^2_{J_2}\mathrm{Ad}_S\,\dot{\mathrm{q}}_2 \\ {}^3\phi_3 &= {}^3_2\mathrm{Ad}\,{}^2\phi_2 + {}^3_{J_3}\mathrm{Ad}_S\,\dot{\mathrm{q}}_3 \\ &= {}^3_2\mathrm{Ad}\left({}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S\,\dot{\mathrm{q}}_1 + {}^2_{J_2}\mathrm{Ad}_S\,\dot{\mathrm{q}}_2\right) + {}^3_{J_3}\mathrm{Ad}_S\,\dot{\mathrm{q}}_3 \\ &= {}^3_2\mathrm{Ad}\,{}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S\,\dot{\mathrm{q}}_1 + {}^3_2\mathrm{Ad}\,{}^2_{J_2}\mathrm{Ad}_S\,\dot{\mathrm{q}}_2 + {}^3_{J_3}\mathrm{Ad}_S\,\dot{\mathrm{q}}_3 \end{aligned} \tag{2.13} $$

The pattern here is that initially, ${}^1\phi_1$ is just a function of $\dot{\mathrm{q}}_1$, but as we traverse the tree, ${}^i\phi_i$ becomes a function of all the ancestors of $i$. For a serial chain, this implies a lower triangular matrix.

$$ \underbrace{\begin{pmatrix} {}^1\phi_1 \\ {}^2\phi_2 \\ {}^3\phi_3 \end{pmatrix}}_{\dot{\mathrm{q}}_m} = \underbrace{\begin{pmatrix} {}^1_{J_1}\mathrm{Ad}_S & 0 & 0 \\ {}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S & {}^2_{J_2}\mathrm{Ad}_S & 0 \\ {}^3_2\mathrm{Ad}\,{}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S & {}^3_2\mathrm{Ad}\,{}^2_{J_2}\mathrm{Ad}_S & {}^3_{J_3}\mathrm{Ad}_S \end{pmatrix}}_{\mathrm{J}_{mr}} \underbrace{\begin{pmatrix} \dot{\mathrm{q}}_1 \\ \dot{\mathrm{q}}_2 \\ \dot{\mathrm{q}}_3 \end{pmatrix}}_{\dot{\mathrm{q}}_r}. \tag{2.14} $$

Note the recursive structure here. To fill a matrix element to the left of the diagonal, we take the element above and premultiply it by ${}^i_{i-1}\mathrm{Ad}$. As we iterate over all the columns to the left of the diagonal, what we are doing is that we are backtracing the ancestors all the way to the root. Note that for a general tree structure, we cannot assume that the row directly above belongs to the parent. Instead, the parent is on some row above the current row. The pseudocode of the Jacobian filling function is given in Alg. 1. This function must be called on the joints in tree traversal order, starting from the root. This function takes advantage of the recursive structure of the tree hierarchy—as we traverse through the ancestors, we use the products already computed by the ancestors. Since this matrix has $O(n^2)$ elements, it takes $O(n^2)$ time to fill, even with this recursive structure. If we only need the product of this matrix with a vector (§2.10), we only need $O(n)$ time, a strategy taken by the recursive forward dynamics algorithm [Featherstone 1983; Kim 2012; Park et al. 1995].

## 2.2 Jacobian Time Derivative

As we saw in Eq. (2.2), we require the time derivative of J, which in turn requires the time derivative of the adjoint, $\dot{\mathrm{Ad}}$. For the diagonal terms, the derivative is

$$ \dot{\mathrm{J}}(i,i) = {}^i_{J_i}\mathrm{Ad}\,\dot{S}, \tag{2.15} $$

**Algorithm 1** Filling the Jacobian matrix

```
J(i, i) = {}^{i}_{J_i}Ad S                          ▷ Diagonal element
ancestor = parent
while ancestor != null do
    J(i, a) = {}^{i}_{p}Ad J(p, a)                  ▷ Off-diagonal element
    ancestor = ancestor's parent
end while
```

which is 0 for revolute joints, since $S$ is constant. (It isn't 0 for some other types of joints.) For off-diagonal terms, recall that we have the recurrence relation $\mathsf{J}(i,a) = {}^{i}_{p}\mathrm{Ad}\,\mathsf{J}(p,a)$, where $p$ is the parent of $i$, and $a$ is an ancestor of $i$. From this, we see that the derivative is

$$\dot{\mathsf{J}}(i,a) = {}^{i}_{p}\dot{\mathrm{Ad}}\,\mathsf{J}(p,a) + {}^{i}_{p}\mathrm{Ad}\,\dot{\mathsf{J}}(p,a). \tag{2.16}$$

To compute ${}^{i}_{p}\dot{\mathrm{Ad}}$, we use Eq. (1.18) and the identity for taking the derivative of the matrix inverse: $\dot{A}^{-1} = -A^{-1}\dot{A}A^{-1}$.

$$\begin{aligned} {}^{i}_{p}\dot{\mathrm{Ad}} &= \frac{d}{dt}\left({}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad}\right) \\ &= {}^{i}_{0}\dot{\mathrm{Ad}}\,{}^{0}_{p}\mathrm{Ad} + {}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\dot{\mathrm{Ad}} \\ &= {}^{0}_{i}\dot{\mathrm{Ad}}^{-1}\,{}^{0}_{p}\mathrm{Ad} + {}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\dot{\mathrm{Ad}} \\ &= -{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{i}\dot{\mathrm{Ad}}\,{}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\mathrm{Ad} + {}^{i}_{0}\mathrm{Ad}\,{}^{0}_{p}\dot{\mathrm{Ad}}. \end{aligned} \tag{2.17}$$

The pseudocode for constructing J and $\dot{\mathsf{J}}$ is given in Alg. 2. The function is called in a forward traversal order, starting from the root. In this ordering, the parent is guaranteed to be processed before its children.

**Algorithm 2** Filling the Jacobian matrix and its time derivative

```
while forward traversal do
    J(i, i) = {}^{i}_{J_i}Ad S                      ▷ Diagonal element
    J̇(i, i) = {}^{i}_{J_i}Ad Ṡ                      ▷ Diagonal element
    ancestor = parent
    while ancestor != null do
        J(i, a) = {}^{i}_{p}Ad J(p, a)              ▷ Off-diagonal element
        J̇(i, a) = {}^{i}_{p}Ȧd J(p, a) + {}^{i}_{p}Ad J̇(p, a)   ▷ Off-diagonal element
        ancestor = ancestor's parent
    end while
end while
```



### 2.3 RedMax Dynamics

Now that we have both $\mathsf{J}_{mr}$ and $\dot{\mathsf{J}}_{mr}$, we can finally form the reduced equations of motion. Combining Eq. (1.21) and Eq. (2.2), we have

$$\begin{aligned} \mathsf{M}_m\left(\mathsf{J}_{mr}\,\ddot{\mathsf{q}}_r + \dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r\right) &= \mathsf{f}_m \\ \mathsf{M}_m\,\mathsf{J}_{mr}\ddot{\mathsf{q}}_r &= \mathsf{f}_m - \mathsf{M}_m\,\dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r \\ \left(\mathsf{J}_{mr}^{\top}\mathsf{M}_m\,\mathsf{J}_{mr}\right)\ddot{\mathsf{q}}_r &= \mathsf{J}_{mr}^{\top}\left(\mathsf{f}_m - \mathsf{M}_m\,\dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r\right) \\ \mathsf{M}_r\ddot{\mathsf{q}}_r &= \mathsf{f}_r, \end{aligned} \tag{2.18}$$

where the reduced mass matrix, $\mathsf{M}_r = \mathsf{J}_{mr}^\top \mathsf{M}_m \mathsf{J}_{mr}$, and the reduced force vector, $\mathsf{f}_r = \mathsf{J}_{mr}^\top \left(\mathsf{f}_m - \mathsf{M}_m \dot{\mathsf{J}}_{mr}\, \dot{\mathsf{q}}_r\right)$, are much smaller than their maximal counterparts (1/6 the size for revolute joints). Furthermore, we don't require constraints, since the Jacobians automatically take care of constraints. The last term, $-\mathsf{J}_{mr}^\top \mathsf{M}_m \dot{\mathsf{J}}_{mr}\, \dot{\mathsf{q}}_r$, is the extra *quadratic velocity vector* that results due to the change of coordinates [Shabana 2013].

Let's first try the simple Euler integration scheme from §1.8. The acceleration in Eq. (2.18) is discretized as

$$\ddot{\mathsf{q}}_r = \frac{\dot{\mathsf{q}}_r^{(k+1)} - \dot{\mathsf{q}}_r^{(k)}}{h}, \tag{2.19}$$

which results in

$$\mathsf{M}_r \dot{\mathsf{q}}_r^{(k+1)} = \mathsf{M}_r \dot{\mathsf{q}}_r^{(k)} + h\mathsf{f}_r. \tag{2.20}$$

This is a small, dense linear system that gives the new reduced velocities, $\dot{\mathsf{q}}_r^{(k+1)}$. If desired, the maximal velocities can be computed using the Jacobian. The reduced positions are integrated as $\mathsf{q}_r{}^{(k+1)} = \mathsf{q}_r{}^{(k)} + h\dot{\mathsf{q}}_r^{(k+1)}$.

Often it is advantageous to use a more sophisticated integrator, such as MATLAB's `ode45` integrator, which allows adaptive time steps. To use these general integrators, we must convert a 2nd order ODE into a system of 1st order ODEs, by stacking the positions and velocities together.

$$\frac{d}{dt}\begin{pmatrix}\mathsf{q}_r \\ \dot{\mathsf{q}}_r\end{pmatrix} = \begin{pmatrix}\dot{\mathsf{q}}_r \\ \mathsf{M}_r^{-1}\mathsf{f}_r\end{pmatrix}. \tag{2.21}$$

An adaptive integrator is much more stable for rigid body dynamics because it takes small time steps as needed. This is important especially if there is no damping, since even a simple two-link system can result in chaotic behavior.[6]

With `ode45`, integrating Eq. (2.21) gives numerically the same solution as the recursive forward dynamics method outlined by Kim [2012]. Because we need to invert the reduced mass matrix, our method is $O(n^3)$, whereas recursive forward dynamics is $O(n)$. (Somehow, the linear method automagically computes the product of the reduced mass matrix with the right-hand-side!) Our method, however, is much simpler to implement, easier to understand, and easier to combine with deformable object simulations (*e.g.*, FEM).

## 2.4 Other Joint Types

These are based on the source code by Kim [2012].[7]

*2.4.1 Fixed Joint.* A fixed joint is used for rigidly attaching two bodies together. Recall that the joint transform is defined wrt the parent joint:

$${}_j^{j-1}\mathsf{E} = {}_j^{j-1}\mathsf{E}_0\, \mathsf{Q}_j(\mathsf{q}_j), \tag{2.22}$$

where ${}_j^{j-1}\mathsf{E}_0$ is the initial transform (often a translation), and $\mathsf{Q}_j(\mathsf{q}_j)$ is the transform that actually applies the degrees of freedom of that joint. For a fixed joint, $\mathsf{q}_j = \varnothing$, and $\mathsf{Q}_j(\mathsf{q}_j)$ is simply the $4 \times 4$ identity matrix. The joint Jacobian, $S$, is an empty $6 \times 0$ matrix.

*2.4.2 Prismatic Joint.* A prismatic joint allows one degree of translational freedom. Let $\boldsymbol{a}$ represent the axis along which the joint is able to translate. Then

$$\mathsf{Q}_j(\mathsf{q}_j) = \begin{pmatrix} I & \boldsymbol{a}\mathsf{q}_j \\ 0 & 1 \end{pmatrix}, \tag{2.23}$$

[6] See a video of a "double pendulum" here: https://youtu.be/U39RMUzCjiU.

[7] GEAR: Geometric Engine for Articulated Rigid-body simulation https://github.com/junggon/gear

which is a $4 \times 4$ translation matrix. The corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 \\ \boldsymbol{a} \end{pmatrix} \in \mathbb{R}^{6\times 1}. \tag{2.24}$$

*2.4.3 Planar Joint.* A planar joint allows translation in two directions. We assume that the joint is oriented so that the allowed motion is in the X-Y plane. Then

$$\mathsf{Q}_j(\mathsf{q}_j) = \begin{pmatrix} I & 0 & \mathsf{q}_j \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.25}$$

which is again a $4 \times 4$ translation matrix. The corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{6\times 2}. \tag{2.26}$$

*2.4.4 Translational Joint.* A translational joint allows full translation (but no rotation).

$$\mathsf{Q}_j(\mathsf{q}_j) = \begin{pmatrix} I & \mathsf{q}_j \\ 0 & 1 \end{pmatrix}, \tag{2.27}$$

and the corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 \\ I \end{pmatrix} \in \mathbb{R}^{6\times 3}. \tag{2.28}$$

*2.4.5 Spherical Joint.* Representing 3D rotation with reduced coordinates is nontrivial. With any 3-parameter representation, there will be a singularity somewhere. With more than 3 parameters, we require constraints, which we will get to in §2.8. One option for a 3-parameter rotation representation is Euler angles. Once we choose a sequence of axes to rotate by (*e.g.,* XZX), we can multiply out the 3 rotation matrices and obtain a single rotation matrix parameterized by the 3 angles.[8]

Recall from Eq. (2.10) that for a revolute joint, $S \in \mathbb{R}^{6\times 1}$, because $\mathsf{q} \in \mathbb{R}$. For a spherical joint parameterized by Euler angles, $S \in \mathbb{R}^{6\times 3}$, and $\mathsf{q} \in \mathbb{R}^3$. Intuitively, each column of $S$ is the derivative of $\mathsf{E}$ wrt $\mathsf{q}$, expressed in local coordinates. In other words, each column $i$ of $S$ is defined as:

$$[S_i] \equiv \mathsf{E}^{-1} \frac{d\mathsf{E}}{d\mathsf{q}_i}, \tag{2.29}$$

where the bracket operator is from Eq. (1.6). (Note the similarity to Eq. (1.8).) By "unbracketing" this LHS matrix, we obtain the $i^{th}$ column of $S$. We can simplify this a little bit because only rotations are involved for a spherical joint. Since $\mathsf{E}$ is a rotation matrix for a spherical joint, we can instead write

$$[S_i] \equiv \mathsf{R}^\top \frac{d\mathsf{R}}{d\mathsf{q}}, \tag{2.30}$$

where the bracket operator corresponds only to the rotational part, as in Eq. (1.7).

[8] Formulas on Wikipedia: https://en.wikipedia.org/wiki/Euler_angles.

Let's use XZX Euler angles as a concrete example. The corresponding rotation matrix is:

$$\mathsf{R} = \begin{pmatrix} c_2 & -s_2c_3 & s_2s_3 \\ c_1s_2 & c_1c_2c_3 - s_1s_3 & -s_1c_3 - c_1c_2s_3 \\ s_1s_2 & c_1s_3 + s_1c_2c_3 & c_1c_3 - s_1c_2s_3 \end{pmatrix}, \tag{2.31}$$

where $c_1 = \cos(q_1), c_2 = \cos(q_2)$, etc. Q is then

$$\mathsf{Q} = \begin{pmatrix} \mathsf{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.32}$$

Since $\mathsf{q} = (q_1\ q_2\ q_3)^\top \in \mathbb{R}^3$, we take the derivative separately three times to get the three columns of $S$. To get the 1st column of $S$, we take the derivative of R wrt $q_1$ and premultiply by $\mathsf{R}^\top$. After lots of cancellations, the resulting product is a skew symmetric matrix:

$$\mathsf{R}^\top \frac{d\mathsf{R}}{dq_1} = \begin{pmatrix} 0 & -s_2s_3 & -c_3s_2 \\ s_2s_3 & 0 & -c_2 \\ c_3s_2 & c_2 & 0 \end{pmatrix}. \tag{2.33}$$

If we "unbracket" this $3 \times 3$ skew symmetric matrix, we obtain a $3 \times 1$ vector. This forms the top three rows of the 1st column of $S$. The bottom three rows are zero, because translations are not parameterized by a spherical joint. Repeating for the 2nd and 3rd rows, we obtain the final form of $S$ for a spherical joint *parameterized by XZX Euler angles*:

$$S = \begin{pmatrix} c_2 & 0 & 1 \\ -c_3s_2 & s_3 & 0 \\ s_2s_3 & c_3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{2.34}$$

For this joint Jacobian, the time derivative, $\dot{S}$, is nonzero, and is needed for the computation of $\dot{\mathsf{J}}$:

$$\dot{S} = \begin{pmatrix} -s_2\dot{q}_2 & 0 & 0 \\ s_3s_2\dot{q}_3 - c_3c_2\dot{q}_2 & c_3\dot{q}_3 & 0 \\ c_2s_3\dot{q}_2 + s_2c_3\dot{q}_3 & -s_3\dot{q}_3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{2.35}$$

*2.4.6 Universal Joint.* A universal joint allows bending in X and Y but no twisting along Z. We start with the rotation matrix corresponding to the XYZ Euler angles:

$$\mathsf{R} = \begin{pmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + s_1s_2c_3 & c_1c_3 - s_1s_2s_3 & -s_1c_2 \\ s_1s_3 - c_1s_2c_3 & s_1c_3 + c_1s_2s_3 & c_1c_2 \end{pmatrix}, \tag{2.36}$$

where $c_1 = \cos(q_1)$, $c_2 = \cos(q_2)$, etc. We then fix the third angle at 0, so that $c_3 = 1$ and $s_3 = 0$. This gives us

$$\mathsf{R} = \begin{pmatrix} c_2 & 0 & s_2 \\ s_1 s_2 & c_1 & -s_1 c_2 \\ -c_1 s_2 & s_1 & c_1 c_2 \end{pmatrix}. \tag{2.37}$$

Q is then

$$\mathsf{Q} = \begin{pmatrix} \mathsf{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.38}$$

The joint Jacobian, $S$, is going to be a $6 \times 2$ matrix. As with the spherical joint, to get the 1st column of $S$, we take the derivative of R wrt $\mathsf{q}_1$ and premultiply by $\mathsf{R}^\top$. After some cancellations, we get a skew symmetric matrix, from which the angular elements are extracted into the first column of $S$. We repeat this for the second column, and the resulting matrix is

$$S = \begin{pmatrix} c_2 & 0 \\ 0 & 1 \\ s_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{2.39}$$

The time derivative of the joint Jacobian is

$$\dot{S} = \begin{pmatrix} -s_2 \dot{q}_2 & 0 \\ 0 & 0 \\ c_2 \dot{q}_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{2.40}$$

*2.4.7 Revolute joint.* Because revolute joints are one of the simplest and most useful joints, we used it as an introductory example in §2.1. Here, we replicate the derivations for completeness. We will assume that the joint allows bending along the Z axis.

$$\mathsf{Q}(\mathsf{q}) = \exp([S\mathsf{q}]) = \begin{pmatrix} \cos(\mathsf{q}) & -\sin(\mathsf{q}) & 0 & 0 \\ \sin(\mathsf{q}) & \cos(\mathsf{q}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad S = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2.41}$$

*2.4.8 Spherical Joint with Exponential Coordinates.* No matter which 3-parameter representation we choose for a spherical joint, there is going to be a singularity somewhere. We could alternatively use a quaternion, but then we would need to add a constraint to keep the quaternion be of unit length. With Euler angles, to stay away from singularities, we need to switch the coordinate chart on the fly (*e.g.,* between ZYX and ZYZ). With exponential coordinates [Gallego and Yezzi 2015; Grassia 1998], we also need to reparameterize, but we do not need to keep track of the coordinate chart.

Let $\mathrm{q} \in \mathrm{so}(3)$ (can also be thought of as $\mathbb{R}^3$) be the DOF of the spherical joint. Recall that every rotation matrix can be expressed as a matrix exponential of a skew symmetric matrix, and Q is then

$$\mathrm{R} = \exp([\mathrm{q}]), \quad \mathrm{Q} = \begin{pmatrix} \mathrm{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.42}$$

The joint Jacobian, $S$, is computed using the derivative formula described by Gallego and Yezzi [2015]. The derivative of R wrt q is a $3 \times 3 \times 3$ tensor, where each $3 \times 3$ slice is given by

$$\frac{\partial \mathrm{R}}{\partial q_i} = \frac{q_i[\mathrm{q}] + [[\mathrm{q}](I - \mathrm{R})\boldsymbol{e}_i]}{\mathrm{q}^\top \mathrm{q}} \mathrm{R}, \tag{2.43}$$

where $\boldsymbol{e}_i$ is the $i^{th}$ standard basis in $\mathbb{R}^3$, and $I$ is the identity matrix. If $\|\mathrm{q}\| < \varepsilon$, then we must take the limit as $\mathrm{q} \to 0$, which gives us $\mathrm{R} = I$, and $\partial \mathrm{R}/\partial q_i = [\boldsymbol{e}_i]$. Each column of the joint Jacobian is then

$$[S_i] = \mathrm{R}^\top \frac{\partial \mathrm{R}}{\partial q_i}. \tag{2.44}$$

The bracket around $S_i$ implies that we need to unbracket the RHS to get each column of $S$. By contracting the $3 \times 3 \times 3$ tensor $\partial \mathrm{R}/\partial \mathrm{q}$ by $\dot{\mathrm{q}}$, we can compute the time derivative of the rotation matrix, $\dot{\mathrm{R}}$, which is needed for $\dot{S}$:

$$\dot{\mathrm{R}} = \sum_i \frac{\partial \mathrm{R}}{\partial q_i} \dot{q}_i. \tag{2.45}$$

To aid in the derivation of $\dot{S}$, we first partition the derivative as

$$\frac{\partial \mathrm{R}}{\partial q_i} = A_i \mathrm{R}, \quad A_i = (B_i + C_i)d, \quad B_i = q_i[\mathrm{q}], \quad C_i = [[\mathrm{q}](I - \mathrm{R})\boldsymbol{e}_i], \quad d = \frac{1}{\mathrm{q}^\top \mathrm{q}}. \tag{2.46}$$

Then each column of $\dot{S}$ can be expressed as

$$\begin{aligned} \left[\dot{S}_i\right] &= \dot{\mathrm{R}}^\top A_i \mathrm{R} + \mathrm{R}^\top \dot{A}_i \mathrm{R} + \mathrm{R}^\top A_i \dot{\mathrm{R}} \\ \dot{A}_i &= \left(\dot{B}_i + \dot{C}_i\right) d + (B_i + C_i)\, \dot{d} \\ \dot{B}_i &= \dot{\mathrm{q}}_i[\mathrm{q}] + \mathrm{q}_i[\dot{\mathrm{q}}] \\ \dot{C}_i &= \left[[\dot{\mathrm{q}}](I - \mathrm{R})\boldsymbol{e}_i - [\mathrm{q}]\dot{\mathrm{R}}\boldsymbol{e}_i\right] \\ \dot{d} &= \frac{-2\mathrm{q}^\top \dot{\mathrm{q}}}{(\mathrm{q}^\top \mathrm{q})^2}. \end{aligned} \tag{2.47}$$

Quoting Grassia [1998], "The exponential map has singularities on the spheres of radius $2n\pi$ (for $n = 1, 2, 3, \ldots$). This makes sense, since a rotation of $2\pi$ about any axis is equivalent to no rotation at all—the entire shell of points $2\pi$ distant from the origin (and 4, etc.) collapses to the identity in SO(3)." They then show that a good way to avoid singularities is to check if $\|\mathrm{q}\|$ is close to $2\pi$ and if so, reparameterize as $\mathrm{q} = (1 - 2\pi/\|\mathrm{q}\|)\mathrm{q}$. Whenever q is reparameterized, we must also update $\dot{\mathrm{q}}$, $S$, and $\dot{S}$. To do so, we first recompute $S$ with Eq. (2.44) using the reparameterized q. Then, we can compute the new velocities as $\dot{\mathrm{q}} = S^{-1} S_{\mathrm{prev}} \dot{\mathrm{q}}_{\mathrm{prev}}$. Finally, we can compute $\dot{S}$ using Eq. (2.47) with the new values of q and $\dot{\mathrm{q}}$.

*2.4.9 Composite Joint.* In a composite joint, two joints are composed together: $\mathrm{Q} = \mathrm{Q}_1 \mathrm{Q}_2$. This can be interpreted as a chaining of two joints, with a massless body in between, with 1 as a parent of 2. The corresponding joint Jacobian is

$$S = \begin{pmatrix} {}^2_1\mathrm{Ad}\, S_1 & S_2 \end{pmatrix} \in \mathbb{R}^{6 \times (n_1 + n_2)}, \tag{2.48}$$

where $n_1$ and $n_2$ are the number of DOFs of joints 1 and 2, respectively, and ${}^2_1\mathrm{Ad}$ is the $6 \times 6$ adjoint matrix that transforms from joint 1's coordinate space to joint 2's coordinate space. The time derivative of the right term, $S_2$, is simply $\dot{S}_2$, which is computed by joint 2. The time derivative of the left term is

$$\frac{d}{dt}\left({}^2_1\mathrm{Ad}\, S_1\right) = {}^2_1\dot{\mathrm{Ad}}\, S_1 + {}^2_1\mathrm{Ad}\, \dot{S}_1. \tag{2.49}$$

To compute ${}^2_1\dot{\mathrm{Ad}}$, note that joint 2 stores its transform wrt joint 1 (child wrt parent), which is ${}^1_2\mathrm{Ad}$. The transform of the parent wrt to the child involves the inverse, and so we have

$$\begin{aligned} {}^2_1\dot{\mathrm{Ad}} &= -{}^2_1\mathrm{Ad}\, {}^1_2\dot{\mathrm{Ad}}\, {}^2_1\mathrm{Ad} && \text{Using the identity for the derivative of the inverse} \\ &= -{}^2_1\mathrm{Ad}\, {}^1_2\mathrm{Ad}\, \mathrm{ad}\left({}^2\phi_2\right)\, {}^2_1\mathrm{Ad} && \text{Using Eq. (1.19)} \\ &= -\mathrm{ad}\left(S_2\dot{\mathrm{q}}_2\right)\, {}^2_1\mathrm{Ad}. \end{aligned} \tag{2.50}$$

The twist of joint 2, ${}^2\phi_2$, is the spatial velocity of 2 wrt 1, which is the product $S_2\dot{\mathrm{q}}_2$. For example, if joint 2 is a translational joint, then

$$S_2\dot{\mathrm{q}}_2 = \begin{pmatrix} 0 \\ \dot{\boldsymbol{q}}_2 \end{pmatrix} \in \mathbb{R}^6, \tag{2.51}$$

which is a translation-only twist. Combining Eq. (2.48), Eq. (2.49), and Eq. (2.50), the time derivative of $S$ is, therefore,

$$\dot{S} = \begin{pmatrix} -\mathrm{ad}\left(S_2\dot{\mathrm{q}}_2\right)\, {}^2_1\mathrm{Ad}\, S_1 + {}^2_1\mathrm{Ad}\, \dot{S}_1 & \dot{S}_2 \end{pmatrix}. \tag{2.52}$$

Composite joints can be chained together. For example, a composite joint of three joints can be expressed as $\mathrm{Q} = \mathrm{Q}_1(\mathrm{Q}_2\mathrm{Q}_3)$.

*2.4.10 2D Free Joint.* A 2D free joint is a joint that is completely unconstrained in 2D. This is useful if we want a structure that is not affixed to the ground, in which case the root joint will be implemented as a free joint. To implement a 2D free joint, we concatenate a X-Y planar joint and a Z revolute joint together into a composite joint:

$$\mathrm{Q} = \mathrm{Q}_1\mathrm{Q}_2, \tag{2.53}$$

where $\mathrm{Q}_1 = \mathrm{Q}_{\text{planar}}$ and $\mathrm{Q}_2 = \mathrm{Q}_{\text{revolute}}$. Their corresponding joint Jacobians are:

$$S_1 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad S_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2.54}$$

and $\dot{S} = 0$ for both. After some simplification, the Jacobian for the 2D free joint is then

$$S = \begin{pmatrix} {}^2_1\mathrm{Ad}\, S_1 & S_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \mathrm{Q}_2^{xx} & \mathrm{Q}_2^{yx} & 0 \\ \mathrm{Q}_2^{xy} & \mathrm{Q}_2^{yy} & 0 \\ \mathrm{Q}_2^{xz} & \mathrm{Q}_2^{yz} & 0 \end{pmatrix}. \tag{2.55}$$

The bottom three rows of the first column contain the 1st row of the $\mathrm{Q}_2$ matrix, and the bottom three rows of the second column contain the 2nd row of the $\mathrm{Q}_2$ matrix. The time derivative of $S$ can be simplified as follows:

$$\dot{S} = \begin{pmatrix} -\mathrm{ad}\left(S_2\dot{\mathrm{q}}_2\right) {}^2_1\mathrm{Ad}\, S_1 + {}^2_1\mathrm{Ad}\, \cancelto{0}{\dot{S}_1} & \cancelto{0}{\dot{S}_2} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \dot{\mathrm{q}}_2\mathrm{Q}_2^{xy} & \dot{\mathrm{q}}_2\mathrm{Q}_2^{yy} & 0 \\ -\dot{\mathrm{q}}_2\mathrm{Q}_2^{xx} & -\dot{\mathrm{q}}_2\mathrm{Q}_2^{xy} & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{2.56}$$

where $\dot{\mathrm{q}}_2$ is the velocity of the revolute joint.

*2.4.11 3D Free Joint.* A 3D free joint is a joint that is completely unconstrained in 3D. This is useful if we want a structure that is not affixed to the ground, in which case the root joint will be implemented as a free joint. To implement a 3D free joint, we concatenate a spherical joint and a translational joint together into a composite joint:

$$\mathrm{Q} = \mathrm{Q}_1\mathrm{Q}_2, \tag{2.57}$$

where $\mathrm{Q}_1 = \mathrm{Q}_{\text{spherical}}$ and $\mathrm{Q}_2 = \mathrm{Q}_{\text{translational}}$. The corresponding joint Jacobian is

$$S = \begin{pmatrix} \hat{S}_1 & 0 \\ -[\boldsymbol{q}_2]\hat{S}_1 & I \end{pmatrix} \in \mathbb{R}^{6\times 6}, \tag{2.58}$$

where $\hat{S}_1$ is the top three rows of $S_1$, and $\boldsymbol{q}_2 \in \mathbb{R}^3$ is the translational DOF of joint 2. The time derivative of the Jacobian is

$$\dot{S} = \begin{pmatrix} \hat{\dot{S}}_1 & 0 \\ -[\dot{\boldsymbol{q}}_2]\hat{S}_1 - [\boldsymbol{q}_2]\hat{\dot{S}}_1 & 0 \end{pmatrix}, \tag{2.59}$$

where $\hat{\dot{S}}_1$ is the top three rows of $\dot{S}_1$, and $\dot{\boldsymbol{q}}_2$ is the (translational) velocity of joint 2.

*2.4.12 Spline Curve Joint.* With RedMax, it is easy to include more advanced joints, such as the Spline Joint by Lee and Terzopoulos [2008]. We'll start by reviewing some basic spline concepts. For concreteness, we'll be using uniform cubic B-spline curves.

Let $C \in \mathbb{R}^{3\times 4}$ be the matrix of 4 consecutive control points:

$$C = \begin{pmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \boldsymbol{c}_3 & \boldsymbol{c}_4 \end{pmatrix}, \tag{2.60}$$

and let $B \in \mathbb{R}^{4\times4}$ be the cubic B-spline basis matrix:

$$B = \frac{1}{6}\begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.61}$$

Then the spline position at $q \in [0, 1]$ can be written as

$$\boldsymbol{x}(q) = CB\vec{q}, \quad \vec{q} = \begin{pmatrix} 1 \\ q \\ q^2 \\ q^3 \end{pmatrix}. \tag{2.62}$$

Other types of splines can be swapped in by replacing the basis matrix, $B$. If there are more than 4 control points, then the matrix $C$ needs to be updated so that the appropriate 4 control points make up the 4 columns of the matrix, and the spline parameter, $q$, must always be mapped to be between 0 and 1.

We can expand Eq. (2.62) in terms of the control points, $\boldsymbol{c}_i$:

$$\boldsymbol{x}(q) = \boldsymbol{c}_1 B_1(q) + \boldsymbol{c}_2 B_2(q) + \boldsymbol{c}_3 B_3(q) + \boldsymbol{c}_4 B_4(q), \tag{2.63}$$

where the basis function, $B_i(q)$, is the product of the $i^{th}$ row of $B$ and $\vec{q}$.

The spline joint uses the cumulative form of basis functions, introduced by Kim et al. [1995]:

$$\boldsymbol{x}(q) = \boldsymbol{c}_1 \tilde{B}_1(q) + \Delta\boldsymbol{c}_2 \tilde{B}_2(q) + \Delta\boldsymbol{c}_3 \tilde{B}_3(q) + \Delta\boldsymbol{c}_4 \tilde{B}_4(q), \tag{2.64}$$

where the control point differences are computed as $\Delta\boldsymbol{c}_i = \boldsymbol{c}_i - \boldsymbol{c}_{i-1}$. By equating Eq. (2.63) and Eq. (2.64), the cumulative basis functions, $\tilde{B}_i(q)$, are:

$$\begin{aligned} \tilde{B}_4(q) &= B_4(q) \\ \tilde{B}_3(q) &= B_3(q) + B_4(q) \\ \tilde{B}_2(q) &= B_2(q) + B_3(q) + B_4(q) \\ \tilde{B}_1(q) &= B_1(q) + B_2(q) + B_3(q) + B_4(q) = 1. \end{aligned} \tag{2.65}$$

The derivatives, $\tilde{B}_i'(q)$ and $\tilde{B}_i''(q)$, are computed by differentiating $\vec{q}$:

$$B'(q) = \frac{1}{6}\begin{pmatrix} -3 & 3 & -1 \\ 0 & -6 & 3 \\ 3 & 3 & -3 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 2q \\ 3q^2 \end{pmatrix}, \quad B''(q) = \frac{1}{6}\begin{pmatrix} 3 & -1 \\ -6 & 3 \\ 3 & -3 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 2 \\ 6q \end{pmatrix}, \tag{2.66}$$

where we have removed the zero entries from $\vec{q}'$ and $\vec{q}''$, and the corresponding columns from $B$.

With the spline joint, instead of control *points* $\boldsymbol{c}_i \in \mathbb{R}^3$, we have control *frames* $\mathrm{C}_i \in \mathrm{SE}(3)$. We use the cumulative form, Eq. (2.64), but instead of subtracting to get the control point differences, we use the matrix logarithm to get the control frame differences. Following Eq. (2.64), the joint matrix can be expressed using products of exponentials instead of additions:

$$\mathrm{Q(q)} = \mathrm{C}_1 \exp\left(\Delta\mathrm{C}_2 \tilde{B}_2(\mathrm{q})\right) \exp\left(\Delta\mathrm{C}_3 \tilde{B}_3(\mathrm{q})\right) \exp\left(\Delta\mathrm{C}_4 \tilde{B}_4(\mathrm{q})\right), \tag{2.67}$$

where the control frame differences are computed using logarithms: $\Delta \mathsf{C}_i = \log(\mathsf{C}_{i-1}^{-1}\mathsf{C}_i)$.

The recursive method for computing the corresponding joint Jacobian, $S = [\mathsf{Q}^{-1}(\partial \mathsf{Q}/\partial \mathsf{q})]$, and Hessian, $\partial S/\partial \mathsf{q}$, are given in the appendix of the spline joints paper [Lee and Terzopoulos 2008], which we reproduce in Alg. 3 for reference. Once we compute $\partial S/\partial \mathsf{q}$, $\dot{S}$ can be computed using the chain rule: $\dot{S} = (\partial S/\partial \mathsf{q})\dot{\mathsf{q}}$.

**Algorithm 3** Cubic Spline Joint transform, Jacobian, and Hessian

$\mathsf{Q} = \mathsf{C}_1 \exp(\Delta \mathsf{C}_2 \tilde{B}_2(\mathsf{q}))$
$S = \Delta \mathsf{C}_2 \tilde{B}_2'(\mathsf{q})$
$\partial S/\partial \mathsf{q} = \Delta \mathsf{C}_2 \tilde{B}_2''(\mathsf{q})$
**for** $i = 3, 4$ **do**
  $\mathsf{Q}_i = \exp(\Delta \mathsf{C}_i \tilde{B}_i(\mathsf{q}))$
  $\mathsf{Q} = \mathsf{Q}\mathsf{Q}_i$
  $\mathsf{Ad}_i = \mathsf{Ad}(\mathsf{Q}_i^{-1})$
  $\mathsf{ad}_i = \mathsf{ad}(S)$
  $S = \Delta \mathsf{C}_i \tilde{B}_i'(\mathsf{q}) + \mathsf{Ad}_i S$
  $\partial S/\partial \mathsf{q} = \Delta \mathsf{C}_i \tilde{B}_i''(\mathsf{q}) + \mathsf{Ad}_i(\partial S/\partial \mathsf{q} + \mathsf{ad}_i \Delta \mathsf{C}_i \tilde{B}_i(\mathsf{q}))$
**end for**

*2.4.13 Spline Surface Joint.* For the spline surface joint (called the multi-DOF spline joint by Lee and Terzopoulos [2008]), we will again use uniform cubic B-splines, and we will limit ourselves to $n = 2$, which means we have a tensor product surface:

$$f(C, q_1, q_2) = \vec{q}_1^{\top} B^{\top} C B \vec{q}_2, \quad \vec{q}_i = \begin{pmatrix} 1 & q_i & q_i^2 & q_i^3 \end{pmatrix}^{\top}, \tag{2.68}$$

where $B$ is the spline basis matrix from Eq. (2.61), and $C$ is the $4 \times 4$ matrix of control values. The derivatives of the tensor product surface are:

$$\begin{aligned} &\frac{\partial f}{\partial q_1} = \vec{q}_1'^{\top} B^{\top} C B \vec{q}_2, \quad \frac{\partial f}{\partial q_2} = \vec{q}_1^{\top} B^{\top} C B \vec{q}_2', \quad \vec{q}_i' = \begin{pmatrix} 0 & 1 & 2q_i & 3q_i^2 \end{pmatrix}^{\top} \\ &\frac{\partial^2 f}{\partial q_1^2} = \vec{q}_1''^{\top} B^{\top} C B \vec{q}_2, \quad \frac{\partial^2 f}{\partial q_2^2} = \vec{q}_1^{\top} B^{\top} C B \vec{q}_2'', \quad \frac{\partial^2 f}{\partial q_1 q_2} = \frac{\partial^2 f}{\partial q_2 q_1} = \vec{q}_1'^{\top} B^{\top} C B \vec{q}_2', \quad \vec{q}_i'' = \begin{pmatrix} 0 & 0 & 2 & 6q_i \end{pmatrix}^{\top}, \end{aligned} \tag{2.69}$$

In the multi-DOF spline joint, Lee and Terzopoulos [2008] suggest using splines to process the 3 rotational and 3 translational degrees of freedom individually. They also suggest putting the translational basis in front of the rotational basis, so that the resulting transformation matrix behaves more intuitively. (*I.e.*, $\mathsf{E} = \mathsf{TR}$ is more intuitive than $\mathsf{E} = \mathsf{RT}$, because the translation values in $\mathsf{T}$ go directly into the last column of the $\mathsf{E}$ matrix rather than being rotated by $\mathsf{R}$. The 6 basis vectors are then:

$$\hat{e}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \hat{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \quad \hat{e}_4 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_5 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_6 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2.70}$$

The resulting transformation is a spline-weighted product of the matrix exponentials of these basis vectors:

$$\mathrm{Q}(\mathrm{q}) = \prod_{k=1}^{6} \exp\left(\hat{e}_k f(C_k, \mathrm{q})\right), \tag{2.71}$$

where $C_k \in \mathbb{R}^{4\times4}$ is matrix of control values for the $k^{th}$ basis. For example, $C_1$ through $C_3$ are the $x$, $y$, and $z$ positions of the 16 control frames. By multiplying the rotations together, the spline frame acts as XYZ Euler angles, and so Lee and Terzopoulos [2008] warn against gimbal locks. This should not be a problem as long as the rotations are small ($< \pi/4$).

The joint Jacobian, $S \in \mathbb{R}^{6\times2}$, and Hessian, $\partial S/\partial \mathrm{q} \in \mathbb{R}^{6\times2\times2}$, can be computed recursively, as shown in Alg. 4. Once we compute $\partial S/\partial \mathrm{q}$, $\dot{S}$ can be computed using the chain rule: $\dot{S} = (\partial S/\partial \mathrm{q})\dot{\mathrm{q}}$, which is a tensor product. In MATLAB notation, this can be written as `Sdot = dSdq(:,:,1)*qdot(1) + dSdq(:,:,2)*qdot(2)`.

**Algorithm 4** Cubic Spline Surface Joint transform, Jacobian, and Hessian

$\mathrm{Q} = \exp(\hat{e}_1 f(C_1, \mathrm{q}))$
**for** $i = 1, 2$ **do**
  $S_i = \hat{e}_1 \partial f_i(\mathrm{C}_1, \mathrm{q})$ ▷ $S_i \in \mathbb{R}^6$ is the $i^{th}$ column of $S$; $\partial f_i = \frac{\partial f}{\partial q_i}$
  **for** $j = 1, 2$ **do**
    $\partial S_{ij} = \hat{e}_1 \partial^2 f_{ij}(C_1, \mathrm{q})$ ▷ $\partial S_{ij} \in \mathbb{R}^6$ is the $(i,j)^{th}$ column of $\partial S/\partial \mathrm{q}$; $\partial^2 f_{ij} = \frac{\partial^2 f}{\partial q_i q_j}$
  **end for**
**end for**
**for** $k = 2, \cdots, 6$ **do**
  $\mathrm{Q}_k = \exp(\hat{e}_k f(C_k, \mathrm{q}))$
  $\mathrm{Q} = \mathrm{Q}\mathrm{Q}_k$
  $\mathrm{Ad}_k = \mathrm{Ad}(\mathrm{Q}_k^{-1})$
  **for** $i = 1, 2$ **do**
    $\mathrm{ad}_i = \mathrm{ad}(S_i)$
    $S_i = \hat{e}_k \partial f_i(C_k, \mathrm{q}) + \mathrm{Ad}_k S_i$
    **for** $j = 1, 2$ **do**
      $\partial S_{ij} = \hat{e}_k \partial^2 f_{ij}(C_k, \mathrm{q}) + \mathrm{Ad}_k(\partial S_{ij} + \mathrm{ad}_i \hat{e}_k \partial f_j(C_k, \mathrm{q}))$
    **end for**
  **end for**
**end for**



## 2.5 Joint Stiffness and Damping

Adding joint stiffness and damping is much easier in reduced coordinates than in maximal coordinates. We'll use linear stiffness here, but non-linear stiffness can be implemented trivially. The joint torque due to the stiffness of the joint is

$$\tau_{\mathrm{K}} = -K\mathrm{q}_r, \tag{2.72}$$

where $K$ is the scalar stiffness parameter. We assumed here that the rest state of the joint is at $\mathrm{q}_r = 0$, but again, it is trivial to have other values. This joint torque goes into the appropriate rows of the reduced force, $\mathrm{f}_r$, which can simply be added to the reduced equations of motion (Eq. (2.18)):

$$\left(\mathrm{J}_{mr}^{\top} \mathrm{M}_m \mathrm{J}_{mr}\right) \ddot{\mathrm{q}}_r = \mathrm{f}_r + \mathrm{J}_{mr}^{\top} \left(\mathrm{f}_m - \mathrm{M}_m \dot{\mathrm{J}}_{mr} \dot{\mathrm{q}}_r\right). \tag{2.73}$$

Similarly, for joint damping, the torque is

$$\tau_{\mathrm{D}} = -D\dot{\mathrm{q}}_r, \tag{2.74}$$

where $D$ is the scalar damping parameter.

With linearly implicit Euler integration, we evaluate the force at the next time step by expanding around the current time step [Baraff and Witkin 1998]. For the joint stiffness force, we get:

$$\begin{aligned}\tau_{\mathrm{K}}{}^{(k+1)} &= \tau_{\mathrm{K}}{}^{(k)} + \frac{\partial \tau_{\mathrm{K}}}{\partial \mathrm{q}_r}\left(\mathrm{q}_r^{(k+1)} - \mathrm{q}_r^{(k)}\right) \\ &= \tau_{\mathrm{K}}{}^{(k)} - Kh\dot{\mathrm{q}}_r^{(k+1)},\end{aligned} \tag{2.75}$$

since $\dot{\mathrm{q}}_r^{(k+1)} = \left(\mathrm{q}_r^{(k+1)} - \mathrm{q}_r^{(k)}\right)/h$ with implicit Euler. So with linearly implicit Euler, the joint stiffness force gets an extra "implicit" term that goes on the left hand side. Similarly, for the joint damping force, we get:

$$\begin{aligned}\tau_{\mathrm{D}}{}^{(k+1)} &= \tau_{\mathrm{D}}{}^{(k)} + \frac{\partial \tau_{\mathrm{D}}}{\partial \dot{\mathrm{q}}_r}\left(\dot{\mathrm{q}}_r^{(k+1)} - \dot{\mathrm{q}}_r^{(k)}\right) \\ &= \tau_{\mathrm{D}}{}^{(k)} - D\left(\dot{\mathrm{q}}_r^{(k+1)} - \dot{\mathrm{q}}_r^{(k)}\right) \\ &= \tau_{\mathrm{D}}{}^{(k)} - D\dot{\mathrm{q}}_r^{(k+1)} - \tau_{\mathrm{D}}{}^{(k)} \\ &= -D\dot{\mathrm{q}}_r^{(k+1)}.\end{aligned} \tag{2.76}$$

So with linearly implicit Euler, the joint damping force gets an "implicit" term that goes on the left hand side, and completely disappears from the right hand side. By moving all the factors of $\dot{\mathrm{q}}_r^{(k+1)}$ from both forces to the right hand side, we get

$$\left(\mathrm{J}_{mr}^{\top}\mathrm{M}_m\,\mathrm{J}_{mr} + h\mathrm{D}_r + h^2\mathrm{K}_r\right)\dot{\mathrm{q}}_r^{(k+1)} = \left(\mathrm{J}_{mr}^{\top}\mathrm{M}_m\,\mathrm{J}_{mr}\right)\dot{\mathrm{q}}_r^{(k)} + h\left(\mathrm{f}_r^{(k)} + \mathrm{J}_{mr}^{\top}\left(\mathrm{f}_m^{(k)} - \mathrm{M}_m\,\dot{\mathrm{J}}_{mr}\,\dot{\mathrm{q}}_r^{(k)}\right)\right). \tag{2.77}$$

For linear stiffness and linear damping (Eq. (2.72) & Eq. (2.74)),

$$\mathrm{K}_r = -\frac{\partial \tau_{\mathrm{K}}}{\partial \mathrm{q}_r} = \begin{pmatrix} K_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & K_n \end{pmatrix}, \qquad \mathrm{D}_r = -\frac{\partial \tau_{\mathrm{D}}}{\partial \dot{\mathrm{q}}_r} = \begin{pmatrix} D_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & D_n \end{pmatrix}. \tag{2.78}$$

(Note: sometimes people move the negative signs around to get $\left(\mathrm{M} + h\mathrm{D} - h^2\mathrm{K}\right)$ on the left hand side [Baraff and Witkin 1998].) In general, we can combine the linearly implicit terms for both reduced and maximal coordinates:

$$\left(\mathrm{J}_{mr}^{\top}\left(\mathrm{M}_m + h\mathrm{D}_m - h^2\mathrm{K}_m\right)\mathrm{J}_{mr} + h\mathrm{D}_r - h^2\mathrm{K}_r\right)\dot{\mathrm{q}}_r^{(k+1)} = \left(\mathrm{J}_{mr}^{\top}\mathrm{M}_m\,\mathrm{J}_{mr}\right)\dot{\mathrm{q}}_r^{(k)} + h\left(\mathrm{f}_r^{(k)} + \mathrm{J}_{mr}^{\top}\left(\mathrm{f}_m^{(k)} - \mathrm{M}_m\,\dot{\mathrm{J}}_{mr}\,\dot{\mathrm{q}}_r^{(k)}\right)\right). \tag{2.79}$$

## 2.6 Hyper Reduced Coordinates

We can further reduce the degrees of freedom by chaining more Jacobians. For example, let's say we have a chain of rigid bodies connected by revolute joints, and we want the joint angle to be all the same. This can be accomplished by adding constraints as shown in §2.8, but if we use a chained Jacobian, we end up with a single DOF system. The reduced equation of motion from before, written out in full, is

$$\mathrm{J}_{mr}^{\top}\,\mathrm{M}_m\,\mathrm{J}_{mr}\,\ddot{\mathrm{q}}_r = \mathrm{J}_{mr}^{\top}\left(\mathrm{f}_m - \mathrm{M}_m\dot{\mathrm{J}}_{mr}\dot{\mathrm{q}}_r\right), \tag{2.80}$$

where $\mathsf{q}_r$ contains all of the joint angles. We now want to apply another Jacobian, so that these joint angles become the same. This can be expressed using the following relationship:

$$\begin{pmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_n \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \dot{\theta} \qquad \dot{\mathsf{q}}_r = \mathsf{J}_{r\mathrm{R}}\, \dot{\mathsf{q}}_{\mathrm{R}}, \tag{2.81}$$

where $\dot{\mathsf{q}}_{\mathrm{R}}$ represents the new (hyper) reduced coordinates. If we define

$$\mathsf{J}_{m\mathrm{R}} = \mathsf{J}_{mr}\, \mathsf{J}_{r\mathrm{R}}, \quad \dot{\mathsf{J}}_{m\mathrm{R}} = \dot{\mathsf{J}}_{mr}\, \mathsf{J}_{r\mathrm{R}} + \mathsf{J}_{mr}\, \dot{\mathsf{J}}_{r\mathrm{R}}, \tag{2.82}$$

then the hyper reduced equation of motion is

$$\mathsf{J}_{m\mathrm{R}}^{\top} \mathsf{M}_m\, \mathsf{J}_{m\mathrm{R}}\, \ddot{\mathsf{q}}_{\mathrm{R}} = \mathsf{J}_{m\mathrm{R}}^{\top} \left( \mathsf{f}_m - \mathsf{M}_m \dot{\mathsf{J}}_{m\mathrm{R}} \dot{\mathsf{q}}_{\mathrm{R}} \right). \tag{2.83}$$

In the following sections, we use lower cased subscripts (*e.g.*, $\mathsf{J}_{mr}$ instead of $\mathsf{J}_{m\mathrm{R}}$) to slightly lighten the notation, but with the understanding that the reduced coordinates can also be hyper reduced.

## 2.7 Adding Deformable Bodies

One of the strengths of the RedMax algorithm is the ease in which deformable objects (*e.g.*, FEM) can be added. Without loss of generality, we show how this can be done with a mass-spring system. Let a spring be defined by a sequence of nodes connected in series, and let $\boldsymbol{x}$ be the nodal positions. For each node, the kinetic energy and the gravitational potential energy can be expressed as

$$T = \frac{1}{2} m \dot{\boldsymbol{x}}^{\top} \dot{\boldsymbol{x}}, \quad V = -m \boldsymbol{g}^{\top} \boldsymbol{x}, \tag{2.84}$$

where $m$ is the mass of the node. This results in mass matrix $M = mI$ and gravity force $\mathsf{f} = m\boldsymbol{g}$. Between consecutive nodes $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$, the elastic potential energy can be expressed as

$$\begin{aligned} V &= \frac{K}{2} \varepsilon^2 \\ \varepsilon &= \frac{l - L}{L} \\ l &= \|\Delta \boldsymbol{x}\| \\ \Delta \boldsymbol{x} &= \boldsymbol{x}_1 - \boldsymbol{x}_0, \end{aligned} \tag{2.85}$$

where $K$ is the stiffness. The corresponding force is the negative gradient of the energy:

$$\boldsymbol{f}_0 = \frac{K\varepsilon}{L} \frac{\Delta \boldsymbol{x}}{l}, \tag{2.86}$$

and $\boldsymbol{f}_1 = -\boldsymbol{f}_0$. These quantities for all of the springs can be collected into a mass matrix $\mathsf{M}_s$ and a force vector $\mathsf{f}_s$.

We can combine this with RedMax by modifying the Jacobian, whose job is to map reduced coordinates into maximal coordinates. Let $\boldsymbol{x}_f$ and $\boldsymbol{x}_a$ denote the "free" and "attached" vertices of the spring. To attach vertices to the rigid bodies, we work in maximal coordinates. The world velocity of the attached vertex can be expressed as:

$$\dot{\boldsymbol{x}}_a = \underbrace{\mathrm{R}\Gamma\left({}^{i}\boldsymbol{x}_a\right)}_{J_{am}} \phi_i, \tag{2.87}$$

where R is the rotation matrix of the body, ${}^i\boldsymbol{x}_a$ is the position of the attached vertex in body coordinates, and $\Gamma \in \mathbb{R}^{3\times 6}$ is the material Jacobian from Eq. (1.12). The time derivative of this Jacobian, $J_{am}$, is:

$$\dot{J}_{am} = \mathrm{R}\,[\boldsymbol{\omega}_i]\,\Gamma. \tag{2.88}$$

By collecting all attachment Jacobians into a single global matrix, we obtain $\mathrm{J}_{am}$, which transforms maximal velocities of rigid bodies to velocities of attached vertices. Let $\mathrm{q}_f$ and $\mathrm{q}_a$ denote the concatenation of *free* and *attached* vertices. Then we have:

$$\begin{aligned}
\begin{pmatrix}\dot{\mathrm{q}}_m\\ \dot{\mathrm{q}}_a\\ \dot{\mathrm{q}}_f\end{pmatrix} &= \begin{pmatrix}\mathrm{J}_{mr} & 0\\ \mathrm{J}_{am}\mathrm{J}_{mr} & 0\\ 0 & I\end{pmatrix}\begin{pmatrix}\dot{\mathrm{q}}_r\\ \dot{\mathrm{q}}_f\end{pmatrix}\\
\begin{pmatrix}\ddot{\mathrm{q}}_m\\ \ddot{\mathrm{q}}_a\\ \ddot{\mathrm{q}}_f\end{pmatrix} &= \begin{pmatrix}\dot{\mathrm{J}}_{mr} & 0\\ \dot{\mathrm{J}}_{am}\mathrm{J}_{mr}+\mathrm{J}_{am}\dot{\mathrm{J}}_{mr} & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}\dot{\mathrm{q}}_r\\ \dot{\mathrm{q}}_f\end{pmatrix} + \begin{pmatrix}\mathrm{J}_{mr} & 0\\ \mathrm{J}_{am}\mathrm{J}_{mr} & 0\\ 0 & I\end{pmatrix}\begin{pmatrix}\ddot{\mathrm{q}}_r\\ \ddot{\mathrm{q}}_f\end{pmatrix}.
\end{aligned} \tag{2.89}$$

The equations above allow us to express the maximal degrees of freedom as a linear function of the reduced degrees of freedom. By using an identity block to pass through the free vertices of the deformable bodies, we can use them as the reduced DOFs but at the same time create maximal quantities (mass matrix, force vector, etc.) for them, without the hassle of reduced coordinates. Let $\mathrm{q}_{\mathrm{M}}$ be the concatenation of the maximal degrees of freedom: $\mathrm{q}_m$, $\mathrm{q}_a$, and $\mathrm{q}_f$, and let $\mathrm{q}_{\mathrm{R}}$ be the concatenation of the reduced degrees of freedom: $\mathrm{q}_r$ and $\mathrm{q}_f$. Then defining $\mathrm{J}_{\mathrm{MR}}$ and $\dot{\mathrm{J}}_{\mathrm{MR}}$ appropriately, Eq. (2.89) can be expressed compactly as

$$\begin{aligned}
\dot{\mathrm{q}}_{\mathrm{M}} &= \mathrm{J}_{\mathrm{MR}}\dot{\mathrm{q}}_{\mathrm{R}}\\
\ddot{\mathrm{q}}_{\mathrm{M}} &= \dot{\mathrm{J}}_{\mathrm{MR}}\dot{\mathrm{q}}_{\mathrm{R}} + \mathrm{J}_{\mathrm{MR}}\ddot{\mathrm{q}}_{\mathrm{R}}.
\end{aligned} \tag{2.90}$$

We define the maximal mass matrix and the maximal force vector as

$$\mathrm{M}_{\mathrm{M}} = \begin{pmatrix}\mathrm{M}_m & 0 & 0\\ 0 & \mathrm{M}_a & 0\\ 0 & 0 & \mathrm{M}_f\end{pmatrix}, \qquad \mathrm{f}_{\mathrm{M}} = \begin{pmatrix}\mathrm{f}_m\\ \mathrm{f}_a\\ \mathrm{f}_f\end{pmatrix}, \tag{2.91}$$

and the resulting reduced equation of motion is

$$\mathrm{J}_{\mathrm{MR}}^{\top}\,\mathrm{M}_{\mathrm{M}}\,\mathrm{J}_{\mathrm{MR}}\,\ddot{\mathrm{q}}_{\mathrm{R}} = \mathrm{J}_{\mathrm{MR}}^{\top}\left(\mathrm{f}_{\mathrm{M}} - \mathrm{M}_{\mathrm{M}}\,\dot{\mathrm{J}}_{\mathrm{MR}}\,\dot{\mathrm{q}}_{\mathrm{R}}\right), \tag{2.92}$$

where $\mathrm{M}_{\mathrm{R}} = \mathrm{J}_{\mathrm{MR}}^{\top}\,\mathrm{M}_{\mathrm{M}}\,\mathrm{J}_{\mathrm{MR}}$ and $\mathrm{f}_{\mathrm{R}} = \mathrm{J}_{\mathrm{MR}}^{\top}\left(\mathrm{f}_{\mathrm{M}} - \mathrm{M}_{\mathrm{M}}\,\dot{\mathrm{J}}_{\mathrm{MR}}\,\dot{\mathrm{q}}_{\mathrm{R}}\right)$ are the reduced mass matrix and force vector, respectively. In the following sections, we use lower cased subscripts (*e.g.,* $\mathrm{J}_{mr}$ instead of $\mathrm{J}_{\mathrm{MR}}$) to slightly lighten the notation, but it is important to note that "reduced coordinates" can mean rigid bodies *with or without* an attached deformable bodies.

## 2.8 Adding Constraints

The Jacobian-based dynamics (Eq. (2.18)) and recursive forward dynamics [Featherstone 1983; Kim 2012; Park et al. 1995] need constraints to support closed loops. These "loop-closing" constraints are implemented in a similar fashion as the joints in maximal coordinate dynamics (§1.10). We're looking for G such that $\mathrm{G}\Phi = 0$.

For example, let's consider forming a four-bar linkage by adding a loop-closing constraint to a system composed of four bodies in a series. The last body will be attached to the first body using a constraint. The world velocities of these

two bodies, $B$ and $A$, are

$$ {}^0\dot{\boldsymbol{x}}_A = {}^0_A\mathrm{R}\,\Gamma({}^A\boldsymbol{x}_A)\,{}^A\phi_A, \quad {}^0\dot{\boldsymbol{x}}_B = {}^0_B\mathrm{R}\,\Gamma({}^B\boldsymbol{x}_B)\,{}^B\phi_B, \tag{2.93} $$

where ${}^A\boldsymbol{x}_A$ and ${}^B\boldsymbol{x}_B$ are the positions of the constrained point expressed in $A$ and $B$, respectively. We want these two velocities to be equal. We must be careful though, because if we simply form a constraint by equating these two, we get a singular system. To see why, consider the number of degrees of freedom and constraints in the system. Before adding the loop-closing constraint, the four-bar linkage has 3 degrees of freedom. If we add a 3-dimensional constraint, how many actual degrees of freedom are we left with? What would happen if we apply this 3D constraint is that we get a singular matrix with a 1D nullspace that corresponds to the actual, single degree of freedom of a four-bar linkage. Resolving this numerically is rather expensive, but fortunately there is a trivial way to get rid of this nullspace beforehand. If we look at the axis of rotation of the joint attached to $A$, we can obtain the two directions orthonormal to $A$'s hinge axis. ($B$ would work just as well.) Let ${}^0\boldsymbol{a}$ be the axis of rotation in world space. We can create two directions orthonormal to ${}^0\boldsymbol{a}$ as follows (dropping the superscript for brevity).

$$ \begin{aligned} \boldsymbol{v}_1 &= (1\ 0\ 0)^\top \qquad \text{// put 1 in the location with the smallest element in abs}(\boldsymbol{a}) \\ \boldsymbol{v}_2 &= \frac{\boldsymbol{a}\times\boldsymbol{v}_1}{\|\boldsymbol{a}\times\boldsymbol{v}_1\|} \\ \boldsymbol{v}_1 &= \frac{\boldsymbol{v}_2\times\boldsymbol{a}}{\|\boldsymbol{v}_2\times\boldsymbol{a}\|}. \end{aligned} \tag{2.94} $$

Then ${}^0\boldsymbol{v}_1$ and ${}^0\boldsymbol{v}_2$ are both vectors in world space orthogonal to ${}^0\boldsymbol{a}$ and to each other. The constraint we want is that the relative velocity must be equal along these two directions.

$$ \underbrace{\begin{pmatrix} {}^0\boldsymbol{v}_1^\top \\ {}^0\boldsymbol{v}_2^\top \end{pmatrix}\begin{pmatrix} {}^0_A\mathrm{R}\,\Gamma({}^A\boldsymbol{x}_A) & -{}^0_B\mathrm{R}\,\Gamma({}^B\boldsymbol{x}_B) \end{pmatrix}}_{\mathrm{G}_m}\underbrace{\begin{pmatrix} {}^A\phi_A \\ {}^B\phi_B \end{pmatrix}}_{\dot{\mathrm{q}}_m} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{2.95} $$

If we are working at the acceleration level, we need to take the time derivative, like we did for the Jacobian in Eq. (2.2). The constraint on the acceleration is

$$ \mathrm{G}_m\dot{\mathrm{q}}_m = 0 \quad \rightarrow \quad \dot{\mathrm{G}}_m\dot{\mathrm{q}}_m + \mathrm{G}_m\ddot{\mathrm{q}}_m = 0. \tag{2.96} $$

For the $\mathrm{G}_m$ in Eq. (2.95), the only factors that depend on time are the two rotation matrices. Taking their time derivative as in Eq. (1.9),

$$ \dot{\mathrm{G}}_m = \begin{pmatrix} {}^0\boldsymbol{v}_1^\top \\ {}^0\boldsymbol{v}_2^\top \end{pmatrix}\begin{pmatrix} {}^0_A\mathrm{R}\,[{}^A\boldsymbol{\omega}_A]\,\Gamma({}^A\boldsymbol{x}_A) & -{}^0_B\mathrm{R}\,[{}^B\boldsymbol{\omega}_B]\,\Gamma({}^B\boldsymbol{x}_B) \end{pmatrix}. \tag{2.97} $$

$\mathrm{G}_m$ and $\dot{\mathrm{G}}_m$ are both $2\times 12$, and they get placed into global constraint matrices as discussed in §1.10. To apply these constraints, we form a KKT system as in Eq. (1.73). Because the constraint is being applied on the maximal coordinates, $\dot{\mathrm{q}}_m$, we must right-multiply the constraints by $\mathrm{J}_{mr}$ first to convert them to reduced coordinates. The constrained dynamics equation is then

$$ \begin{pmatrix} \mathrm{M}_r & \mathrm{J}_{mr}^\top\,\mathrm{G}_m^\top \\ \mathrm{G}_m\,\mathrm{J}_{mr} & 0 \end{pmatrix}\begin{pmatrix} \ddot{\mathrm{q}}_r \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathrm{f}_r \\ -\left(\dot{\mathrm{G}}_m\,\mathrm{J}_{mr} + \mathrm{G}_m\,\dot{\mathrm{J}}_{mr}\right)\dot{\mathrm{q}}_r \end{pmatrix}. \tag{2.98} $$

If, instead, we are working at the velocity level, the constraint is $G_m J_{mr} \dot{q}_r = 0$, and so we get

$$\begin{pmatrix} M_r & J_{mr}^\top G_m^\top \\ G_m J_{mr} & 0 \end{pmatrix} \begin{pmatrix} \dot{q}_r^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} M_r \dot{q}_r^{(k)} + h f_r{}^{(k)} \\ 0 \end{pmatrix}. \tag{2.99}$$

If, instead, the constraint applies directly to the reduced coordinates rather than maximal coordinates, then the KKT system does not need the $J_{mr}$ factors in the constraints. At the acceleration level,

$$\begin{pmatrix} M_r & G_r^\top \\ G_r & 0 \end{pmatrix} \begin{pmatrix} \ddot{q}_r \\ \lambda \end{pmatrix} = \begin{pmatrix} f_r \\ -\dot{G}_r \dot{q}_r \end{pmatrix}, \tag{2.100}$$

and at the velocity level,

$$\begin{pmatrix} M_r & G_r^\top \\ G_r & 0 \end{pmatrix} \begin{pmatrix} \dot{q}_r^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} M_r \dot{q}_r^{(k)} + h f_r{}^{(k)} \\ 0 \end{pmatrix}. \tag{2.101}$$

Sometimes we may want both types of constraints: those acting on maximal coordinates, and those acting on reduced coordinates. Then substituting

$$\bar{G}_r = \begin{pmatrix} G_r \\ G_m J_{mr} \end{pmatrix}, \quad \dot{\bar{G}}_r = \begin{pmatrix} \dot{G}_r \\ \dot{G}_m J_{mr} + G_m \dot{J}_{mr} \end{pmatrix} \tag{2.102}$$

into Eq. (2.100) will automatically apply both types of constraints. When expanded out, the expression turns into:

$$\begin{pmatrix} M_r & G_r^\top & J_{mr}^\top G_m^\top \\ G_r & 0 & 0 \\ G_m J_{mr} & 0 & 0 \end{pmatrix} \begin{pmatrix} \ddot{q}_r \\ \lambda_r \\ \lambda_m \end{pmatrix} = \begin{pmatrix} f_r \\ -\dot{G}_r \dot{q}_r \\ -\left(\dot{G}_m J_{mr} + G_m \dot{J}_{mr}\right) \dot{q}_r \end{pmatrix}. \tag{2.103}$$

Quadratic programs for inequality constraints can be constructed similarly. In the most general case, we have inequality and equality constraints on both maximal and reduced coordinates, represented by constraint matrices $C_m$, $C_r$, $G_m$, and $G_r$, respectively. In addition to Eq. (2.102) and let

$$\bar{C}_r = \begin{pmatrix} C_r \\ C_m J_{mr} \end{pmatrix}, \quad \dot{\bar{C}}_r = \begin{pmatrix} \dot{C}_r \\ \dot{C}_m J_{mr} + C_m \dot{J}_{mr} \end{pmatrix}. \tag{2.104}$$

Then the resulting quadratic program is

$$\begin{aligned} \underset{\ddot{q}_r}{\text{minimize}} \quad & \frac{1}{2} \ddot{q}_r^\top M_r \ddot{q}_r - \ddot{q}_r^\top f_r \\ \text{subject to} \quad & \bar{C}_r \ddot{q}_r \geq -\dot{\bar{C}}_r \dot{q}_r \\ & \bar{G}_r \ddot{q}_r = -\dot{\bar{G}}_r \dot{q}_r. \end{aligned} \tag{2.105}$$

## 2.9 Hybrid Dynamics

In forward dynamics, we compute the accelerations given the forces, and in inverse dynamics, we compute the forces given the accelerations. In the RedMax formulation, it is easy to mix these two into "hybrid" dynamics. Let $p$ indicate the subset of joints whose motion are prescribed. Then we can apply an equality constraint on the prescribed accelerations: ${}^p G_r \, \ddot{q}_r = {}^p \ddot{q}_r$, where ${}^p G_r$ contains the identity matrix in the appropriate columns so that the prescribed joints will be affected (note ${}^p \dot{G}_r = 0$). The KKT system is then

$$\begin{pmatrix} M_r & \bar{G}_r^\top \\ \bar{G}_r & 0 \end{pmatrix} \begin{pmatrix} \ddot{q}_r \\ \lambda \end{pmatrix} = \begin{pmatrix} f_r \\ {}^p \ddot{q}_r - \dot{\bar{G}}_r \dot{q}_r \end{pmatrix}, \tag{2.106}$$

where we have included the $-\dot{\bar{\mathrm{G}}}_r \dot{\mathrm{q}}_r$ term since other constraints may have non-zero $\dot{\bar{\mathrm{G}}}_r$. The required joint torques can be computed with the resulting Lagrange multiplier: ${}^p\tau = {}^p\mathrm{G}_r^\top {}^p\lambda$, where ${}^p\mathrm{G}_r$ and ${}^p\lambda$ are the appropriate rows and columns of $\mathrm{G}_r$ and $\lambda$, respectively. At the velocity level,

$$\begin{pmatrix} \mathrm{M}_r & \bar{\mathrm{G}}_r^\top \\ \bar{\mathrm{G}}_r & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathrm{q}}_r^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathrm{M}_r \dot{\mathrm{q}}_r^{(k)} + h\mathrm{f}_r^{(k)} \\ h\,{}^p\ddot{\mathrm{q}}_r + {}^p\mathrm{G}_r \dot{\mathrm{q}}_r^{(k)} \end{pmatrix}. \tag{2.107}$$

The 2nd row of the KKT system, $\bar{\mathrm{G}}_r \dot{\mathrm{q}}_r = h\,{}^p\ddot{\mathrm{q}}_r + {}^p\mathrm{G}_r \dot{\mathrm{q}}_r^{(k)}$, contains an extra term on the right hand side because the joint acceleration, rather than velocity, is being prescribed. It is also possible to prescribe the velocity by replacing the 2nd row with ${}^p\mathrm{G}_r\,\dot{\mathrm{q}}_r = {}^p\dot{\mathrm{q}}_r$.

## 2.10 Jacobian Products

Forming the Jacobian is $O(n^2)$, but computing the products $\mathrm{y} = \mathrm{J}\mathrm{x}$, $\mathrm{z} = \dot{\mathrm{J}}\mathrm{x}$, and $\mathrm{x} = \mathrm{J}^\top \mathrm{y}$ can all be done in $O(n)$. This is useful for obtaining linear time recursive hybrid dynamics with constraints. Here, $j$ refers to joint frame, $p$ refers to the parent of $j$, $c$ refers to a child of $j$, and $i$ refers to the body owned by $j$. Forward traversal starts from the root, and backward traversal starts from a leaf. In forward traversal, the parent is processed before its children, and in backward traversal, all children are processed before their parent.

Recall the structure of the Jacobian, as we saw in Eq. (2.14):

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} {}^1_{J_1}\mathrm{Ad}_S & 0 & 0 \\ {}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S & {}^2_{J_2}\mathrm{Ad}_S & 0 \\ {}^3_2\mathrm{Ad}\,{}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S & {}^3_2\mathrm{Ad}\,{}^2_{J_2}\mathrm{Ad}_S & {}^3_{J_3}\mathrm{Ad}_S \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}. \tag{2.108}$$

Given this triangular structure, to compute $\mathrm{y} = \mathrm{J}\mathrm{x}$, we start with the top row: $y_1 = {}^1_{J_1}\mathrm{Ad}_S\, x_1$. For the 2nd row, note that the parent (*i.e.,* row 1) already computed a subexpression that we need, and so $y_2 = {}^2_1\mathrm{Ad}\, y_1 + {}^2_{J_2}\mathrm{Ad}_S\, x_2$. For the 3rd row, note again that we have already computed much of the subexpression, since

$$\begin{aligned} y_3 &= {}^3_2\mathrm{Ad}\,{}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S\, x_1 + {}^3_2\mathrm{Ad}\,{}^2_{J_2}\mathrm{Ad}_S\, x_2 + {}^3_{J_3}\mathrm{Ad}_S\, x_3 \\ &= {}^3_2\mathrm{Ad}\,\left({}^2_1\mathrm{Ad}\,{}^1_{J_1}\mathrm{Ad}_S\, x_1 + {}^2_{J_2}\mathrm{Ad}_S\, x_2\right) + {}^3_{J_3}\mathrm{Ad}_S\, x_3 \\ &= {}^3_2\mathrm{Ad}\,\left({}^2_1\mathrm{Ad}\, y_1 + {}^2_{J_2}\mathrm{Ad}_S\, x_2\right) + {}^3_{J_3}\mathrm{Ad}_S\, x_3 \\ &= {}^3_2\mathrm{Ad}\, y_2 + {}^3_{J_3}\mathrm{Ad}_S\, x_3. \end{aligned} \tag{2.109}$$

So the recursive expression is $y_i = {}^i_p\mathrm{Ad}\, y_p + {}^i_j\mathrm{Ad}_S\, x_j$. (Keep in mind that $i$ and $j$ are almost interchangeable. For each joint $j$, there is a corresponding body $i$, and vice-versa.) For a general tree structure, we need to do a forward traversal starting from the root, so that the parents are processed before their children. The resulting algorithm is shown in Alg. 5.

Computing the product with the Jacobian transpose is slightly more involved.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} {}^1_{J_1}\mathrm{Ad}_S^\top & {}^1_{J_1}\mathrm{Ad}_S^\top\,{}^2_1\mathrm{Ad}^\top & {}^1_{J_1}\mathrm{Ad}_S^\top\,{}^2_1\mathrm{Ad}^\top\,{}^3_2\mathrm{Ad}^\top & {}^1_{J_1}\mathrm{Ad}_S^\top\,{}^2_1\mathrm{Ad}^\top\,{}^3_2\mathrm{Ad}^\top\,{}^4_3\mathrm{Ad}^\top \\ 0 & {}^2_{J_2}\mathrm{Ad}_S^\top & {}^2_{J_2}\mathrm{Ad}_S^\top\,{}^3_2\mathrm{Ad}^\top & {}^2_{J_2}\mathrm{Ad}_S^\top\,{}^3_2\mathrm{Ad}^\top\,{}^4_3\mathrm{Ad}^\top \\ 0 & 0 & {}^3_{J_3}\mathrm{Ad}_S^\top & {}^3_{J_3}\mathrm{Ad}_S^\top\,{}^4_3\mathrm{Ad}^\top \\ 0 & 0 & 0 & {}^4_{J_4}\mathrm{Ad}_S^\top \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}. \tag{2.110}$$

**Algorithm 5** Compute products $y = \mathrm{J}x$ and $z = \dot{\mathrm{J}}x$

**while** forward traversal **do**
  $y(i) = {}^{i}_{j}\mathrm{Ad}\, S\, x(j)$
  $z(i) = {}^{i}_{j}\mathrm{Ad}\, \dot{S}\, x(j)$
  **if** parent != null **then**
    $y(i)$ += ${}^{i}_{p}\mathrm{Ad}\, y(p)$
    $z(i)$ += ${}^{i}_{p}\mathrm{Ad}\, z(p) + {}^{i}_{p}\dot{\mathrm{Ad}}\, y(p)$
  **end if**
**end while**

Given this upper triangular structure, we start at the bottom and go up (*i.e.*, backward traversal from leaf to root).

$$
\begin{aligned}
x_4 &= {}^{4}_{J_4}\mathrm{Ad}_S^{\top}\, y_4 \\
x_3 &= {}^{3}_{J_3}\mathrm{Ad}_S^{\top}\, y_3 + {}^{3}_{J_3}\mathrm{Ad}_S^{\top}\, {}^{4}_{3}\mathrm{Ad}^{\top}\, y_4 \\
&= {}^{3}_{J_3}\mathrm{Ad}_S^{\top}\left(y_3 + {}^{4}_{3}\mathrm{Ad}^{\top}\, y_4\right) \\
x_2 &= {}^{2}_{J_2}\mathrm{Ad}_S^{\top}\, y_2 + {}^{2}_{J_2}\mathrm{Ad}_S^{\top}\, {}^{3}_{2}\mathrm{Ad}^{\top}\, y_3 + {}^{2}_{J_2}\mathrm{Ad}_S^{\top}\, {}^{3}_{2}\mathrm{Ad}^{\top}\, {}^{4}_{3}\mathrm{Ad}^{\top}\, y_4 \\
&= {}^{2}_{J_2}\mathrm{Ad}_S^{\top}\left(y_2 + \left({}^{3}_{2}\mathrm{Ad}^{\top}\, y_3 + {}^{4}_{3}\mathrm{Ad}^{\top}\, y_4\right)\right) \\
x_1 &= {}^{1}_{J_1}\mathrm{Ad}_S^{\top}\, y_1 + {}^{1}_{J_1}\mathrm{Ad}_S^{\top}\, {}^{2}_{1}\mathrm{Ad}^{\top}\, y_2 + {}^{1}_{J_1}\mathrm{Ad}_S^{\top}\, {}^{2}_{1}\mathrm{Ad}^{\top}\, {}^{3}_{2}\mathrm{Ad}^{\top}\, y_3 + {}^{1}_{J_1}\mathrm{Ad}_S^{\top}\, {}^{2}_{1}\mathrm{Ad}^{\top}\, {}^{3}_{2}\mathrm{Ad}^{\top}\, {}^{4}_{3}\mathrm{Ad}^{\top}\, y_4 \\
&= {}^{1}_{J_1}\mathrm{Ad}_S^{\top}\left(y_1 + \left({}^{2}_{1}\mathrm{Ad}^{\top}\, y_2 + \left({}^{3}_{2}\mathrm{Ad}^{\top}\, y_3 + {}^{4}_{3}\mathrm{Ad}^{\top}\, y_4\right)\right)\right).
\end{aligned}
\tag{2.111}
$$

This can be expressed recursively using a temporary variable $\alpha$, stored for each joint:

$$
\begin{aligned}
x_4 &= {}^{4}_{J_4}\mathrm{Ad}_S^{\top}\,(y_4 + 0), & \alpha_4 &= {}^{4}_{3}\mathrm{Ad}^{\top}\,(y_4 + 0) \\
x_3 &= {}^{3}_{J_3}\mathrm{Ad}_S^{\top}\,(y_3 + \alpha_4), & \alpha_3 &= {}^{3}_{2}\mathrm{Ad}^{\top}\,(y_3 + \alpha_4) \\
x_2 &= {}^{2}_{J_2}\mathrm{Ad}_S^{\top}\,(y_2 + \alpha_3), & \alpha_2 &= {}^{2}_{1}\mathrm{Ad}^{\top}\,(y_2 + \alpha_3) \\
x_1 &= {}^{1}_{J_1}\mathrm{Ad}_S^{\top}\,(y_1 + \alpha_2), & \alpha_1 &= \varnothing.
\end{aligned}
\tag{2.112}
$$

This is implemented in Alg. 6.

**Algorithm 6** Compute product $x = \mathrm{J}^{\top}y$

**while** backward traversal **do**
  $y_i = y(i)$
  **for all** children $c$ **do**
    $y_i$ += $\alpha_c$ ▷ $\alpha$ is a temporary variable stored by each joint
  **end for**
  $\alpha_i = {}^{i}_{p}\mathrm{Ad}^{\top}\, y_i$ ▷ to be used by $i$'s parent later
  $x(j) = S^{\top}\, {}^{i}_{j}\mathrm{Ad}^{\top}\, y_i$
**end while**

## 2.11 Bilateral Staggered Projections

The original Staggered Projections (SP) algorithm was developed for solids undergoing unilateral contact constraints with friction [Kaufman et al. 2008]. SP is shown in Alg. 7, slightly modified to match our notation. Since SP was designed

for maximal rigid bodies, we remove the $m$ and $r$ (maximal & reduced) subscripts for clarity. There are two quadratic programs (QP) that are solved iteratively: contact and friction. Let $\dot{\mathsf{q}}^{\text{unc}} = \dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}\mathsf{f}$ be the unconstrained velocity. The contact QP can then be written as:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \frac{1}{2}\alpha^\top \mathsf{N}\mathsf{M}^{-1}\mathsf{N}^\top\alpha - \alpha^\top \mathsf{N}(\dot{\mathsf{q}}^{\text{unc}} + h\mathsf{M}^{-1}\mathsf{f}_\beta) \\ \text{subject to} \quad & \alpha \geq 0, \end{aligned} \tag{2.113}$$

where $\alpha$ is the contact impulse, $\mathsf{N}$ is the contact normal matrix, $\mathsf{M}$ is the maximal mass matrix, $\mathsf{f}$ is the maximal force, and $\mathsf{f}_\beta$ is the frictional force, which is initially zero. After solving for $\alpha$, we can compute the contact force as $\mathsf{f}_\alpha = -\mathsf{N}^\top\alpha/h$. The frictional QP is:

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & \frac{1}{2}\beta^\top \mathsf{T}\mathsf{M}^{-1}\mathsf{T}^\top\beta - \beta^\top \mathsf{T}(\dot{\mathsf{q}}^{\text{unc}} + h\mathsf{M}^{-1}\mathsf{f}_\alpha) \\ \text{subject to} \quad & -\mu\alpha \leq \beta \leq \mu\alpha, \end{aligned} \tag{2.114}$$

where $\beta$ is the frictional impulse, $\mathsf{T}$ is the contact tangent matrix, and $\mu \geq 0$ is the coefficient of friction. The box constraints can only accommodate a four-sided friction cone—if needed, we can rewrite this constraint to give us a better approximation or switch to a quadratically constrained quadratic program. After solving for $\beta$, we can compute the frictional force as $\mathsf{f}_\beta = -\mathsf{T}^\top\beta/h$. In most simulations, the convergence rate can be improved by caching the frictional force, $\mathsf{f}_\beta$, and warm-starting SP with this cached value at every time step.

**Algorithm 7** Staggered Projections

Fill mass matrix $\mathsf{M}$
$\mathsf{f}_\beta = 0$
**while** simulating **do**
  Fill force vector $\mathsf{f}$
  Fill contact normal matrix $\mathsf{N}$
  Fill contact tangent matrix $\mathsf{T}$
  $\mathsf{f}_\alpha^0 = 0$
  $\dot{\mathsf{q}}^{\text{unc}} = \dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}\mathsf{f}$
  **while** true **do**
    // CONTACT
    Solve contact QP (2.113) for $\alpha$
    $\mathsf{f}_\alpha = -\mathsf{N}^\top\alpha/h$
    // CONVERGENCE CHECK
    **if** $\|\mathsf{f}_\alpha - \mathsf{f}_\alpha^0\|_{\mathsf{M}^{-1}} \leq \epsilon$ **or** max iterations **then**
      **break**
    **end if**
    $\mathsf{f}_\alpha^0 = \mathsf{f}_\alpha$
    // FRICTION
    Solve friction QP (2.114) for $\beta$
    $\mathsf{f}_\beta = -\mathsf{T}^\top\beta/h$
  **end while**
  $\dot{\mathsf{q}} = \dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}(\mathsf{f} + \mathsf{f}_\alpha + \mathsf{f}_\beta)$
**end while**

We can extend SP by taking advantage of the bilateral constraints present in articulated rigid body dynamics. The resulting algorithm, which we call Bilateral Staggered Projections (BISP), is an order of magnitude faster than SP and

can also be combined with SP for handling external frictional contacts, such as between a body and the environment [to be confirmed].

The first advantage with bilateral contacts is that we do not need collision detection. With BISP, for each joint type (*e.g.*, revolute, spherical, prismatic), we use implicit contacts at pre-determined positions around the joint. For example, for a revolute joint, we assume that the joint geometry is a cylinder, and we populate the two ends of the cylinder with contact points. By changing the parameters of this cylinder, we get different frictional effects.

In reduced coordinates, we do not need a contact QP, since the reduced equations of motion automatically give us reduced velocities that satisfy the joint constraints. For frictional contact, however, we need access to the Lagrange multipliers of the contact constraints. In BISP, we compute these multipliers by first solving for the equivalent constraint forces that would produce the same constrained motion as the one generated by reduced coordinates. This can be done by comparing the velocity generated by the reduced solve against the velocity generated by an unconstrained solve. In other words, we can rearrange the constrained equations of motion to solve for the constraint forces:

$$\begin{aligned} \mathsf{M}\dot{\mathsf{q}} + \mathsf{N}^\top \alpha &= \mathsf{M}\dot{\mathsf{q}}^{\text{prev}} + h\mathsf{f} \\ \underbrace{\dot{\mathsf{q}}}_{\dot{\mathsf{q}}^{\text{con}}} + \mathsf{M}^{-1}\mathsf{N}^\top \alpha &= \underbrace{\dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}\mathsf{f}}_{\dot{\mathsf{q}}^{\text{unc}}}. \end{aligned} \tag{2.115}$$

The right hand side is the unconstrained velocity, obtained by ignoring the constraint force. The first term on the left hand side is the solution to the constrained equations of motion, which we call $\dot{\mathsf{q}}^{\text{con}}$ to be explicit. We can rearrange the second term on the left hand side to obtain the expression for the constraint force, $\mathsf{f}_\alpha = -\mathsf{N}^\top \alpha / h$:

$$\begin{aligned} \mathsf{M}^{-1}\mathsf{N}^\top \alpha &= \dot{\mathsf{q}}^{\text{unc}} - \dot{\mathsf{q}}^{\text{con}} \\ \mathsf{N}^\top \alpha &= \mathsf{M}\left(\dot{\mathsf{q}}^{\text{unc}} - \dot{\mathsf{q}}^{\text{con}}\right) \\ \mathsf{f}_\alpha &= \frac{1}{h}\mathsf{M}\left(\dot{\mathsf{q}}^{\text{con}} - \dot{\mathsf{q}}^{\text{unc}}\right). \end{aligned} \tag{2.116}$$

With BISP, the contact solve is replaced by a reduced solve. As shown in Eq. (2.116), we can compute the constraint force, $\mathsf{f}_\alpha$, by comparing the constrained velocity to the unconstrained velocity. Also, as in SP, the current friction force must be taken into account when computing the constrained and unconstrained velocities. In the equations below, since we now must differentiate between reduced and maximal coordinates, we add back the subscripts $m$ and $r$. (The contact and friction forces, $\mathsf{f}_\alpha$ and $\mathsf{f}_\beta$, are maximal quantities.)

$$\begin{aligned} \dot{\mathsf{q}}_m^{\text{unc}\beta} &= \mathsf{J}_{mr}\dot{\mathsf{q}}_r^{\text{prev}} + h\mathsf{M}_m^{-1}\left(\mathsf{f}_m + \mathsf{f}_\beta\right) \\ \dot{\mathsf{q}}_m^{\text{con}\beta} &= \mathsf{J}_{mr}\left(\dot{\mathsf{q}}_r^{\text{prev}} + h\mathsf{M}_r^{-1}\left(\mathsf{J}_{mr}^\top\left(\mathsf{f}_m + \mathsf{f}_\beta - \mathsf{M}_m\dot{\mathsf{J}}_{mr}\dot{\mathsf{q}}_r^{\text{prev}}\right)\right)\right) \\ \mathsf{f}_\alpha &= \frac{1}{h}\mathsf{M}_m\left(\dot{\mathsf{q}}_m^{\text{con}\beta} - \dot{\mathsf{q}}_m^{\text{unc}\beta}\right). \end{aligned} \tag{2.117}$$

The computed constraint force, $\mathsf{f}_\alpha$, is a global force vector that accounts for all joint reaction forces. Therefore, if we extract a portion of $\mathsf{f}_\alpha$ corresponding to a single body, what we obtain is the sum of all the joint reaction forces acting on that body. To compute the Lagrange multipliers for joint contacts, we first need to isolate the joint reaction force from this sum. Fortunately, this can be done in a linear fashion by traversing the joints backward from the leaf. (Reminder: assuming there are no cycles, there is a one-to-one mapping between joints and bodies. Each body owns a joint that connects the body to the parent body.) For a leaf body, there is only one joint acting on it, and so the portion of $\mathsf{f}_\alpha$ is exactly the required joint reaction force. Since this joint reaction force exerts an equal and opposite force on the

**Algorithm 8** Bilateral Staggered Projections

Fill mass matrix $\mathsf{M}$
$\mathsf{f}_\beta = 0$
**while** simulating **do**
  Fill force vector $\mathsf{f}$
  Fill contact normal matrix $\mathsf{N}$
  Fill contact tangent matrix $\mathsf{T}$
  $\mathsf{f}_\alpha^0 = 0$
  **while** true **do**
    // CONTACT
    Evaluate (2.117) for $\mathsf{f}_\alpha$
    **while** backward traversal **do**
      Distribute $\mathsf{f}_\alpha$ to joint
    **end while**
    **while** parallel traversal **do**
      Locally solve (2.118) for $\alpha$
    **end while**
    // CONVERGENCE CHECK
    **if** $\|\mathsf{f}_\alpha - \mathsf{f}_\alpha^0\|_{\mathsf{M}^{-1}} \leq \epsilon$ **or** max iterations **then**
      **break**
    **end if**
    $\mathsf{f}_\alpha^0 = \mathsf{f}_\alpha$
    // FRICTION
    Solve friction QP (2.114) for $\beta$
    $\mathsf{f}_\beta = -\mathsf{T}^\top \beta / h$
  **end while**
  $\dot{\mathsf{q}}_r = \dot{\mathsf{q}}_r^{\text{prev}} + h\,\mathsf{M}_r^{-1}(\mathsf{f}_r + \mathsf{J}_{mr}^\top(\mathsf{f}_\alpha + \mathsf{f}_\beta))$
**end while**



parent of the leaf, we must subtract this force from the parent's portion of $\mathsf{f}_\alpha$ before continuing the backward traversal. As long as we process all the children before the parent, when we process a body, its portion of $\mathsf{f}_\alpha$ will be exactly the required joint reaction force. While processing a body, before subtracting the equal and opposite force from the parent, we must first transform the force with the adjoint transpose to the parent's frame.

Once we have the joint reaction forces distributed to each joint, we can compute the contact Lagrange multipliers that generate the joint reaction force. This can be done in parallel, since these are local operations performed for each joint independently of each other. For each joint, we solve the following linear system:

$$\alpha = h\left(\mathsf{N}_i \mathsf{M}_i^{-1} \mathsf{N}_i^\top + \epsilon I\right)^{-1}\left(\mathsf{N}_i \mathsf{M}_i^{-1} \mathsf{f}_{\alpha i}\right), \tag{2.118}$$

where the subscript $i$ indicates the blocks corresponding to the $i^{th}$ joint. We do not require $\alpha$ to be positive, since these "contact" constraints are bilateral—they cannot come apart. The regularization term ensures that we obtain the smallest contact impulses. This is critical because otherwise the joint can become arbitrarily tight. For instance, for a revolute joint, setting $\alpha = 1000$ for all contacts will generate the same effective constraint as setting $\alpha = 0.1$.

After the contact impulses, $\alpha$, are computed, the convergence check and the friction solve are the same as in SP.

*2.11.1 BISP with External Constraints.* BISP can also take into account external constraints, such as loop-closing (bilateral) constraints or frictional contact (unilateral) constraints with the environment. BISP (Alg. 8) is modified as follows to take into account these additional constraints.

- Line 10: To compute the contact force, both the unconstrained and constrained velocities must take into account the additional external constraints. The unconstrained velocity is obtained by solving a maximal system with only the external constraints, ignoring the implicit constraints exerted by the joints. The corresponding constrained velocity is obtained by solving a reduced system with the external constraint. The difference between these velocities will give us the joint reaction forces. Thus, instead of Eq. (2.117), we evaluate the following:

$$\begin{aligned}
\dot{\mathsf{q}}_m^{\mathrm{unc}\beta} &= \underset{\dot{\mathsf{q}}_m}{\operatorname{argmin}} \quad \frac{1}{2}\dot{\mathsf{q}}_m^\top \mathsf{M}_m \dot{\mathsf{q}}_m - \dot{\mathsf{q}}_m^\top \left(\mathsf{M}_m \mathsf{J}_{mr} \dot{\mathsf{q}}_r^{\mathrm{prev}} + h\left(\mathsf{f}_m + \mathsf{f}_\beta\right)\right) \\
&\qquad \text{subject to} \quad \mathsf{G}_m \dot{\mathsf{q}}_m = 0 \\
&\qquad\qquad\qquad\quad \mathsf{C}_m \dot{\mathsf{q}}_m \geq 0, \\
\dot{\mathsf{q}}_m^{\mathrm{con}\beta} &= \underset{\dot{\mathsf{q}}_r}{\operatorname{argmin}} \quad \frac{1}{2}\dot{\mathsf{q}}_r^\top \mathsf{M}_r \dot{\mathsf{q}}_r - \dot{\mathsf{q}}_r^\top \left(\mathsf{M}_r \dot{\mathsf{q}}_r^{\mathrm{prev}} + h\mathsf{J}_{mr}^\top \left(\mathsf{f}_m + \mathsf{f}_\beta - \mathsf{M}_m \dot{\mathsf{J}}_{mr} \dot{\mathsf{q}}_r^{\mathrm{prev}}\right)\right) \\
&\qquad \text{subject to} \quad \mathsf{G}_m \mathsf{J}_{mr} \dot{\mathsf{q}}_r = 0 \\
&\qquad\qquad\qquad\quad \mathsf{C}_m \mathsf{J}_{mr} \dot{\mathsf{q}}_r \geq 0, \\
\mathsf{f}_\alpha &= \frac{1}{h}\mathsf{M}_m \left(\mathsf{J}_{mr}\dot{\mathsf{q}}_r^{\mathrm{con}\beta} - \dot{\mathsf{q}}_m^{\mathrm{unc}\beta}\right).
\end{aligned} \tag{2.119}$$

  The loop-closing constraint reaction forces are computed as $\mathsf{f}_\lambda = -\mathsf{J}_{mr}^\top \mathsf{G}_m^\top \lambda / h$, where $\lambda$ is the vector of Lagrange multipliers corresponding to the loop-closing constraints, $\mathsf{G}_m \mathsf{J}_{mr} \dot{\mathsf{q}}_r = 0$, from the minimization for $\dot{\mathsf{q}}_m^{\mathrm{con}\beta}$.
- Line 15: We need to compute the contact forces due to the loop-closing constraints, by again solving a small linear system (2.118). These small linear systems are solved for each joint *and* for each loop-closing constraint.
- Line 26: To compute the final velocity, we solve a quadratic program that takes into account the external constraints:

$$\begin{aligned}
&\underset{\dot{\mathsf{q}}_r}{\text{minimize}} \quad \frac{1}{2}\dot{\mathsf{q}}_r^\top \mathsf{M}_r \dot{\mathsf{q}}_r - \dot{\mathsf{q}}_r^\top \left(\mathsf{M}_r \dot{\mathsf{q}}_r^{\mathrm{prev}} + h\mathsf{J}_{mr}^\top \left(\mathsf{f}_m + \mathsf{f}_\alpha + \mathsf{f}_\beta - \mathsf{M}_m \dot{\mathsf{J}}_{mr} \dot{\mathsf{q}}_r^{\mathrm{prev}}\right)\right) \\
&\text{subject to} \quad \mathsf{G}_m \mathsf{J}_{mr} \dot{\mathsf{q}}_r = 0 \\
&\qquad\qquad\quad \mathsf{C}_m \mathsf{J}_{mr} \dot{\mathsf{q}}_r \geq 0.
\end{aligned} \tag{2.120}$$

## 2.12 Recursive Hybrid Dynamics

Just for reference, we duplicate here the recursive hybrid dynamics algorithm by Kim and Pollard [2011] and Kim [2012], with minor notational changes. Here, $j$ refers to joint frame, $p$ refers to the parent of $j$, $c$ refers to a child of $j$, and $i$ refers to the body owned by $j$.

List of symbols:

- $\mathsf{q}_j, \dot{\mathsf{q}}_j, \ddot{\mathsf{q}}_j \in \mathbb{R}^{n_j}$: generalized position, velocity, acceleration of the joint.
- $\tau_j \in \mathbb{R}^{n_j}$: torque (or generalized force) on joint.
- ${}^p_j\mathsf{E} \in \mathrm{SE}(3)$: Transformation matrix from $p$ to $j$, a function of $\mathsf{q}_j$. *E.g.*, see Eq. (2.5) and Eq. (2.31).
- $S_j = \left({}^p_j\mathsf{E}^{-1}\frac{d\,{}^p_j\mathsf{E}}{d\mathsf{q}_1} \quad \cdots \quad {}^p_j\mathsf{E}^{-1}\frac{d\,{}^p_j\mathsf{E}}{d\mathsf{q}_{n_j}}\right) \in \mathbb{R}^{6\times n_j}$: Jacobian of ${}^p_j\mathsf{E}$, viewed in $j$. *E.g.*, see Eq. (2.5) and Eq. (2.34).

- $\mathsf{V}_j \in \text{se}(3)$: twist at the joint. The twist at the body center is ${}^i\phi_i = {}^i_j\text{Ad}\,\mathsf{V}_j$ and vice-versa.
- $\dot{\mathsf{V}}_j \in \text{se}(3)$: Component-wise time derivative of $\mathsf{V}_j$.
- $\mathsf{M}_j \in \mathbb{R}^{6\times6}$: Body inertia at the joint. Since the joint is not at the body center, it is not diagonal. It can be calculated from body-centered inertia as $\mathsf{M}_j = {}^i_j\text{Ad}^\top \mathsf{M}_i\, {}^i_j\text{Ad}$.
- $\mathsf{F}_j \in \text{dse}(3)$: Generalized force acting on the joint due to the connection to the parent, viewed in $j$. (dse(3) is the space of generalized forces acting on SE(3))
- $\mathsf{F}^{\text{ext}}_j \in \text{dse}(3)$: Generalized external force viewed in $i$. Gravity can be calculated as $\mathsf{F}^{\text{ext}}_j = {}^i_j\text{Ad}^\top\, \mathsf{F}^{\text{grav}}_i$, where $\mathsf{F}^{\text{grav}}_i$ is the force of gravity acting on the body center. (It has zeros in the top three rows and $m\boldsymbol{g}$ in the bottom three rows.)

**Algorithm 9** Recursive Hybrid Dynamics

**while** forward traversal **do**
  ${}^p_j\mathsf{E}$ = function of $\mathsf{q}_j$
  $\mathsf{V}_j = {}^j_p\text{Ad}\,\mathsf{V}_p + S_j\dot{\mathsf{q}}_j$
  $\eta_j = \text{ad}(\mathsf{V}_j)\,S_j\dot{\mathsf{q}}_j + \dot{S}_j\dot{\mathsf{q}}_j$
**end while**
**while** backward traversal **do**
  $\hat{\mathsf{M}}_j = \mathsf{M}_j + \sum_c {}^c_j\text{Ad}^\top\, \Pi_c\, {}^c_j\text{Ad}$
  $\hat{\mathcal{B}}_j = -\text{ad}(\mathsf{V}_j)^\top \mathsf{M}_j\mathsf{V}_j - \mathsf{F}^{\text{ext}}_j + \sum_c {}^c_j\text{Ad}^\top\, \beta_c$
  **if** prescribed acceleration **then**
    $\Pi_j = \hat{\mathsf{M}}_j$
    $\beta_j = \hat{\mathcal{B}}_j + \hat{\mathsf{M}}_j\left(\eta_j + S_j\ddot{\mathsf{q}}_j\right)$
  **else**
    $\Psi_j = (S_j^\top \hat{\mathsf{M}}_j S_j)^{-1}$
    $\Pi_j = \hat{\mathsf{M}}_j - \hat{\mathsf{M}}_j S_j \Psi_j S_j^\top \hat{\mathsf{M}}_j$
    $\beta_j = \hat{\mathcal{B}}_j + \hat{\mathsf{M}}_j\left(\eta_j + S_j\Psi_j\left(\tau_j - S_j^\top\left(\hat{\mathsf{M}}_j\eta_j + \hat{\mathcal{B}}_j\right)\right)\right)$
  **end if**
**end while**
**while** forward traversal **do**
  **if** prescribed acceleration **then**
    $\dot{\mathsf{V}}_j = {}^j_p\text{Ad}\,\dot{\mathsf{V}}_p + S_j\ddot{\mathsf{q}}_j + \eta_j$
    $\mathsf{F}_j = \hat{\mathsf{M}}_j\dot{\mathsf{V}}_j + \hat{\mathcal{B}}_j$
    $\tau_j = S_j^\top \mathsf{F}_j$
  **else**
    $\ddot{\mathsf{q}}_j = \Psi_j\left(\tau_j - S_j^\top\hat{\mathsf{M}}_j\left({}^j_p\text{Ad}\,\dot{\mathsf{V}}_p + \eta_j\right) - S_j^\top\hat{\mathcal{B}}_j\right)$
    $\dot{\mathsf{V}}_j = {}^j_p\text{Ad}\,\dot{\mathsf{V}}_p + S_j\ddot{\mathsf{q}}_j + \eta_j$
    $\mathsf{F}_j = \hat{\mathsf{M}}_j\dot{\mathsf{V}}_j + \hat{\mathcal{B}}_j$
  **end if**
**end while**

The Recursive Hybrid Dynamics algorithm can also be used with constraints. Let the equality and inequality constraints be (as before)

$$\bar{\mathsf{G}}_r = \begin{pmatrix} \mathsf{G}_r \\ \mathsf{G}_m \mathsf{J}_{mr} \end{pmatrix}, \quad \dot{\bar{\mathsf{G}}}_r = \begin{pmatrix} \dot{\mathsf{G}}_r \\ \dot{\mathsf{G}}_m \mathsf{J}_{mr} + \mathsf{G}_m \dot{\mathsf{J}}_{mr} \end{pmatrix}, \quad \bar{\mathsf{C}}_r = \begin{pmatrix} \mathsf{C}_r \\ \mathsf{C}_m \mathsf{J}_{mr} \end{pmatrix}, \quad \dot{\bar{\mathsf{C}}}_r = \begin{pmatrix} \dot{\mathsf{C}}_r \\ \dot{\mathsf{C}}_m \mathsf{J}_{mr} + \mathsf{C}_m \dot{\mathsf{J}}_{mr} \end{pmatrix}. \tag{2.121}$$

Then we can solve the dual problem [Boyd and Vandenberghe 2004] using the fact that the Recursive Hybrid Dynamics algorithm can be used to "invert" or "solve using" the reduced mass matrix. The basic idea is to solve the unconstrained problem first and then to compute the constrained forces required to fix the constraint violations. The computed constrained forces can then be reapplied to obtain the final accelerations that produces feasible motion.

We will first assume that we only have equality constraints. Let $\ddot{\mathsf{q}}_r^*$ be the unconstrained accelerations obtained by ignoring the constraints. The primal problem to compute the change in acceleration to produce feasible motion is:

$$\begin{pmatrix} \mathsf{M}_r & \bar{\mathsf{G}}_r^\top \\ \bar{\mathsf{G}}_r & 0 \end{pmatrix} \begin{pmatrix} \partial\ddot{\mathsf{q}}_r \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ -\dot{\bar{\mathsf{G}}}_r \dot{\mathsf{q}}_r - \bar{\mathsf{G}}_r \ddot{\mathsf{q}}_r^* \end{pmatrix}. \tag{2.122}$$

By applying block Gaussian elimination (also known as Schur complement or Delassus operator), we arrive at the dual problem:

$$\bar{\mathsf{G}}_r \mathsf{M}_r^{-1} \bar{\mathsf{G}}_r^\top \lambda = -\dot{\bar{\mathsf{G}}}_r \dot{\mathsf{q}}_r - \bar{\mathsf{G}}_r \ddot{\mathsf{q}}_r^*, \tag{2.123}$$

which can be solved for the Lagrange multipliers, $\lambda$. To form the left-hand-side matrix, we need the inverse of the reduced mass matrix. Rather than forming this matrix explicitly, we backsolve using the columns of $\bar{\mathsf{G}}_r^\top$ (or equivalently the rows of $\bar{\mathsf{G}}_r$). This can be accomplished by running the Recursive Hybrid Dynamics algorithm with force and momentum terms removed. The computed accelerations then form a column of the inverse product. If there are maximal constraints, $\mathsf{G}_m$, then we also have to take the product of the Jacobian with a vector, since we must backsolve with the columns of $\mathsf{J}_{mr}^\top \mathsf{G}_m^\top$. This means that we take a column of $\mathsf{G}_m$ (or equivalently a row of $\mathsf{G}_m$), multiply by $\mathsf{J}_{mr}$, then backsolve with the resulting product. Both the Jacobian products and the backsolve must be done in *linear time* using the recursive algorithms Alg. 5, Alg. 6, and Alg. 9. The constraint forces to keep the system feasible are then

$$\mathsf{f}_{\text{con}} = \bar{\mathsf{G}}_r^\top \lambda. \tag{2.124}$$

We can then rerun the Recursive Hybrid Dynamics algorithm with $\mathsf{f}_{\text{con}}$ as an external force.

Now we will add inequality constraints. The primal problem for computing the change in acceleration due to constraints is

$$\begin{aligned} \underset{\partial\ddot{\mathsf{q}}_r}{\text{minimize}} \quad & \frac{1}{2} \partial\ddot{\mathsf{q}}_r^\top \mathsf{M}_r \, \partial\ddot{\mathsf{q}}_r \\ \text{subject to} \quad & \bar{\mathsf{C}}_r \, \partial\ddot{\mathsf{q}}_r \geq -\dot{\bar{\mathsf{C}}}_r \dot{\mathsf{q}}_r - \bar{\mathsf{C}}_r \ddot{\mathsf{q}}_r^* \\ & \bar{\mathsf{G}}_r \, \partial\ddot{\mathsf{q}}_r = -\dot{\bar{\mathsf{G}}}_r \dot{\mathsf{q}}_r - \bar{\mathsf{G}}_r \ddot{\mathsf{q}}_r^*. \end{aligned} \tag{2.125}$$

The corresponding dual problem is to solve for the Lagrange multipliers instead. First, let $\bar{\mathsf{A}}_r$ be the combined constraint matrix:

$$\bar{\mathsf{A}}_r = \begin{pmatrix} \bar{\mathsf{C}}_r \\ \bar{\mathsf{G}}_r \end{pmatrix}, \quad \dot{\bar{\mathsf{A}}}_r = \begin{pmatrix} \dot{\bar{\mathsf{C}}}_r \\ \dot{\bar{\mathsf{G}}}_r \end{pmatrix}. \tag{2.126}$$

Then the dual problem is

$$\begin{aligned} \underset{\lambda}{\text{minimize}} \quad & \frac{1}{2}\lambda^\top \bar{\mathsf{A}}_r \mathsf{M}_r^{-1} \bar{\mathsf{A}}_r^\top \lambda + \lambda^\top \left( \dot{\bar{\mathsf{A}}}_r \dot{\mathsf{q}}_r + \bar{\mathsf{A}}_r \ddot{\mathsf{q}}_r^* \right) \\ \text{subject to} \quad & \lambda_\mathsf{C} \leq 0, \end{aligned} \tag{2.127}$$

where $\lambda_\mathsf{C}$ contains the Lagrange multipliers corresponding to the inequality constraints, $\bar{\mathsf{C}}_r$. The Lagrange multipliers corresponding to the equality constraints, $\bar{\mathsf{G}}_r$, are unconstrained.

## 2.13 Inverse Inertia via RFD

We can use the recursive forward dynamics algorithm to compute the inverse inertia product in $O(n)$ time [Drumwright 2012; Kim 2012]. The insight is to recognize that the recursive forward dynamics algorithm solves $\mathsf{M}\ddot{\mathsf{q}} = \mathsf{f}$, which implies that it can compute $y = \mathsf{M}^{-1}x$ by inputting an arbitrary vector $x$ as the generalized force and extracting the solution from the computed acceleration. Therefore, the inverse inertia product can be computed in $O(n)$ time, and the inverse inertia matrix can be formed in $O(n^2)$ time by using the columns of the identity matrix as the right-hand-side vectors. Alg. 10 shows the steps. The first backward traversal loop needs to be run just once as a preprocessing step. The next two loops then must be run for each right-hand-side vector $x$. In the preprocessing loop, we can optionally include joint damping and stiffness for linearly implicit integration. If there are maximal damping and stiffness, we can only include the diagonal terms, since off-diagonal terms break the tree structure of the system.

**Algorithm 10** Inverse reduced inertia product via recursive forward dynamics. Computes $y = \mathsf{M}_r^{-1}x$ in linear time. Alternatively, it can compute $y = (\mathsf{M}_r + \mathsf{J}_{mr}^\top \text{blkdiag}(h\mathsf{D}_m - h^2\mathsf{K}_m)\mathsf{J}_{mr} + h\mathsf{D}_r - h^2\mathsf{K}_r)^{-1}x$ for preconditioning a linearly implicit solver.

// Run this loop once as a preprocessing step
**while** backward traversal **do**
  $\mathsf{P}_j^r = 0$
  $\mathsf{P}_j^m = 0$
  **if** preconditioner **then**
    $\mathsf{P}_j^r = h\mathsf{D}_j^r - h^2\mathsf{K}_j^r$ ▷ Reduced terms
    $\mathsf{P}_j^m = {}_i^j\mathsf{Ad}^\top \, \text{blkdiag}(h\mathsf{D}_i^m - h^2\,\mathsf{K}_i^m)\,{}_j^i\mathsf{Ad}$ ▷ Maximal terms
  **end if**
  $\hat{\mathsf{M}}_j = \mathsf{M}_j + \mathsf{P}_j^m + \sum_c {}_j^c\mathsf{Ad}^\top \Pi_c \, {}_j^c\mathsf{Ad}$
  $\Psi_j = (S_j^\top \hat{\mathsf{M}}_j S_j + \mathsf{P}_j^r)^{-1}$
  $\Pi_j = \hat{\mathsf{M}}_j - \hat{\mathsf{M}}_j S_j \Psi_j S_j^\top \hat{\mathsf{M}}_j$
**end while**

// Run these two loops for each RHS vector $x$
**while** backward traversal **do**
  $\hat{\mathcal{B}}_j = \sum_c {}_j^c\mathsf{Ad}^\top \beta_c$
  $\beta_j = \hat{\mathcal{B}}_j + \hat{\mathsf{M}}_j \left( S_j \Psi_j \left( x_j - S_j^\top \hat{\mathcal{B}}_j \right) \right)$
**end while**
**while** forward traversal **do**
  $y_j = \Psi_j \left( x_j - S_j^\top \hat{\mathsf{M}}_j \, {}_p^j\mathsf{Ad}\, \dot{\mathsf{V}}_p - S_j^\top \hat{\mathcal{B}}_j \right)$
  $\dot{\mathsf{V}}_j = {}_p^j\mathsf{Ad}\, \dot{\mathsf{V}}_p + S_j y_j$
**end while**

## REFERENCES


Mihai Anitescu and Gary D Hart. 2004. A Fixed-point Iteration Approach for Multibody Dynamics with Contact and Small Friction. *MATH. PROG.* 101, 1 (2004), 3–32.

David Baraff. 1996. Linear-time Dynamics Using Lagrange Multipliers. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/237170.237226

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Proc. SIGGRAPH 98, Annual Conference Series*. 43–54.

J. Baumgarte. 1972. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering* 1 (Jun 1972), 1–16.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

M. B. Cline and D. K. Pai. 2003. Post-stabilization for rigid body simulation with contact and constraints. In *Proc. IEEE International Conference on Robotics and Automation*, Vol. 3. 3744–3751.

Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. 2007. OpenSim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering* 54, 11 (2007), 1940–1950.

Evan Drumwright. 2012. Fast dynamic simulation of highly articulated robots with contact via $\Theta(n^2)$ time dense generalized inertia matrix inversion. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 65–76.

Roy Featherstone. 1983. The calculation of robot dynamics using articulated-body inertias. *The International Journal of Robotics Research* 2, 1 (1983), 13–30.

Guillermo Gallego and Anthony Yezzi. 2015. A compact formula for the derivative of a 3-D rotation in exponential coordinates. *Journal of Mathematical Imaging and Vision* 51, 3 (2015), 378–384.

F. Sebastin Grassia. 1998. Practical Parameterization of Rotations Using the Exponential Map. *J. Graph. Tools* 3, 3 (March 1998), 29–48. https://doi.org/10.1080/10867651.1998.10487493

Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai. 2008. Staggered projections for frictional contact in multibody systems. *ACM Trans. Graph.* 27, 5, Article 164 (Dec 2008), 11 pages. Issue 5.

Junggon Kim. 2012. *Lie Group Formulation of Articulated Rigid Body Dynamics*. Technical Report. Carnegie Mellon University.

Junggon Kim and Nancy S Pollard. 2011. Fast simulation of skeleton-driven deformable body characters. *ACM Transactions on Graphics (TOG)* 30, 5 (2011), 121.

Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. 1995. A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 369–376. https://doi.org/10.1145/218380.218486

Sung-Hee Lee and Demetri Terzopoulos. 2008. Spline Joints for Multibody Dynamics. *ACM Trans. Graph.* 27, 3, Article 22 (Aug. 2008), 8 pages. https://doi.org/10.1145/1360612.1360621

Matthew Millard, Thomas Uchida, Ajay Seth, and Scott L Delp. 2013. Flexing computational muscle: modeling and simulation of musculotendon dynamics. *Journal of biomechanical engineering* 135, 2 (2013), 021005.

Brian Mirtich. 1996. Fast and accurate computation of polyhedral mass properties. *Journal of graphics tools* 1, 2 (1996), 31–50.

Richard M. Murray, Zexiang Li, and S. Shankar Sastry. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press.

Frank C Park, James E Bobrow, and Scott R Ploen. 1995. A Lie group formulation of robot dynamics. *The International Journal of Robotics Research* 14, 6 (1995), 609–618.

Jon M Selig. 2004. Lie groups and Lie algebras in robotics. In *Computational Noncommutative Algebra and Applications*. Springer, 101–125.

Ahmed A Shabana. 2013. *Dynamics of multibody systems*. Cambridge university press.