Computing constraint Jacobians

Claude Lacoursière
HPC2N/UMIT
and
Department of Computing Science
Umeå University
SE-901 87, Umeå, Sweden
claude@hpc2n.umu.se

December 1, 2011

1 Introduction

These notes should explain more about how to compute Jacobians for simulating simple multibody systems which are only subject to distance and contact constraints.

2 What's a Jacobian?

Properly speaking, a Jacobian is a *gradient* and for a given scalar function $g(x)$ of a real vector $x \in \mathbb{R}^n$, that's defined as the *row* vector

$$\frac{\partial g}{\partial x} = \nabla g = G = \left[\frac{\partial g}{\partial x_1} \quad \frac{\partial g}{\partial x_2} \quad \ldots, \frac{\partial g}{\partial x_n} \right]. \tag{1}$$

In my convention here, lower case letters are either scalars or vectors, and upper case letters refer to matrices. I will use $g(x)$ for any function, but in particular, the functions that define constraints when $g(x) = 0$.

Consider the simple example where $g(x) = \|x\|$, i.e., the norm of the vector x which can also be written as $\sqrt{x^T x} = \sqrt{x \cdot x}$, where x^T is the transpose of vector x. I write the dot-product of two vectors of compatible dimensions as either $x^T y$ or $x \cdot y$. The convention here is that vectors are always column vectors, and so x^T is a row vector. Using standard matrix-matrix multiplication rules, $x^T x = \sum_{i=1}^{n} x_i^2$, and this is a scalar as it should.

Now, think of x as a scalar and now, by chain rule, we have

$$\frac{\mathrm{d}\sqrt{x^2}}{\mathrm{d}x} = \frac{1}{2\sqrt{x^2}}\frac{\mathrm{d}x^2}{\mathrm{d}x} = \frac{x}{\sqrt{x^2}}. \tag{2}$$

We're almost there. Now, consider that x is an n-dimensional vector and use the chain rule again so that

$$\begin{aligned}\nabla g = \nabla\|x\| = \nabla\sqrt{x^T x} &= \frac{1}{2\sqrt{x^T x}}(\nabla x^T x) \\ &= \frac{1}{\sqrt{x^T x}}x^T = \frac{1}{\|x\|}x^T.\end{aligned} \tag{3}$$

This is just the transpose of the unit vector pointing in the direction of x. The reason to keep the fractions in front is to separate the vectors and the scalars. The transpose has to be there because of the convention for ∇g being a row vector.

Now, if the function $g(x)$ is also a vector of m-dimensions, meaning that

$$g(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{bmatrix} \tag{4}$$

we can still compute ∇g_i for each components. Since that gives us a row vector, the complete Jacobian $\nabla g = G$ is now an $m \times n$ matrix, namely

$$\nabla g = G = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \cdots & \frac{\partial g_m}{\partial x_n}. \end{bmatrix} \tag{5}$$

That looks daunting at first but it's going to be very simple. Recall that in our cases, the function $g(x)$ is a constraint, and that usually involves only two bodies. So, if x is the vector of coordinates of the entire system, we'll have $\partial g_i/\partial x_j = 0$ unless j is the coordinate of one body constrained by g_i.

Also, don't worry too much about matrices and storage, since you only need the nonzero components. In the case of particles or rigid body constraints, you'll need $2m$ small matrices to store a Jacobian and you can put that in your data structure for a constraint, or keep pointers to a storage area if you like. That's what we do in the AgX code to keep everything nice and tight, i.e., we use structs of arrays instead of arrays of structs.

3 Point particles

First consider a particle at position x and the constraint $g(x) = \|x\| - l$. That's a simple pendulum hooked to the origin by a string of length l. The Jacobian for that

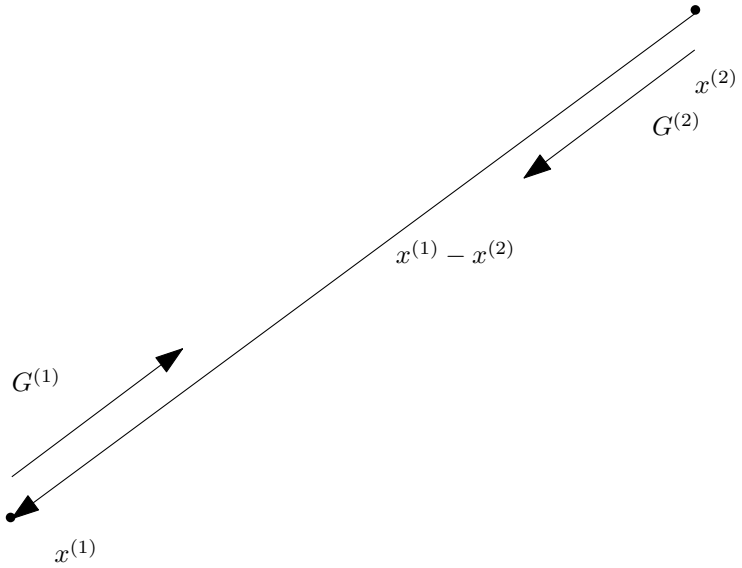

Figure 1: A distance constraint between two particles. The Jacobians here are simply normals aligned along the center line between the particles. What else could it be!

is given by Eqn. (3). As the Jacobian points in the direction of fastest increase, we can move along $-(1/\|x\|)x^T$ to move radially toward the origin. If you remember the definition of a constraint force as $G^T\lambda$, you can guess that $\lambda < 0$ is the force needed to keep the particle from flying away.

Now consider two particles $x^{(1)}, x^{(2)}$ and the length constraint $g(x) = \|x^{(1)} - x^{(2)}\| - l = 0$, where l is a constant. Here, I am writing x as the set of generalized coordinates so that

$$x = \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} \tag{6}$$

where $x_j^{(1)}, j = 1, 2, 3$ are the coordinates of the first particle, and same for the second.

For the case of particles, we have three dimensions for positions and three dimensions for velocities. You'll see later that this does not work exactly the same way for rigid bodies because of the quaternion representation of the rotations. But don't worry yet.

So, going back to the definition in Eqn. (1), we need to get all partial derivatives of $g(x)$ with respect to all coordinates. But you might already guess that there is a lot of symmetry between what's going on with $x^{(1)}$ and $x^{(2)}$. To make the chain rule clear

and easy to use here, write $r = x^{(1)} - x^{(2)}$. That's just a 3D vector now and we can get the partials easily

$$\begin{aligned} \frac{\partial r}{\partial x^{(1)}} &= \frac{\partial x^{(1)} - x^{(2)}}{\partial x^{(1)}} = I \\ \frac{\partial r}{\partial x^{(2)}} &= \frac{\partial x^{(1)} - x^{(2)}}{\partial x^{(2)}} = -I \end{aligned} \tag{7}$$

where I is the 3×3 identity matrix. Recall now that r is a 3D vector so that $\partial r / \partial x^{(i)}, i = 1, 2$ must be a 3×3 matrix as shown already. So now, using the chain rule, we have

$$\begin{aligned} \frac{\partial g}{\partial x^{(1)}} &= \frac{1}{2\|r\|} \frac{\partial r^T r}{\partial x^{(1)}} = \frac{1}{\|r\|} r^T \frac{\partial r}{\partial x^{(1)}} \\ &= \frac{1}{\|r\|} r^T I = \frac{1}{\|r\|} r^T. \end{aligned} \tag{8}$$

Clearly, for particle 2, we get the opposite sign

$$\frac{\partial g}{\partial x^{(1)}} = -\frac{1}{\|r\|} r^T. \tag{9}$$

Recall now that the gradient of a function is the direction of fastest increase and clearly now, this shows that moving the particles in the central direction $\pm 1/\|r\| r$ changes the distance as fast as possible. Which is obvious.

What happens if we have a long chain of n particles each linked to its neighbor with a distance constraint? That gives us $m = n - 1$ constraint equations and so we need to compute the beast of a matrix described in Eqn. (5). If you look at that matrix, the particles identify the columns, and the constraints are on the rows. Let's label the constraints from 1 to $n - 1$ so that constraint g_i is the one that keeps particles $x^{(i)}$ and $x^{(i+1)}$ at the given distance l_ii. So now, for a given constraint $g_i(x)$, we will have $\partial g_i / \partial x^{(j)} = 0$ unless $j = i$ or $j = i - 1$. Given what was computed for the two particle case, we'll then have

$$\frac{\partial g_i}{\partial x^{(i)}} = \frac{1}{\|r_{i,i+1}\|}(x^{(i)} - x^{(i+1)}) = \frac{1}{\|r_{ii+1}\|} r_{i,i+1}, \text{ where } r_{i,j} = r_{ij} = x^{(i)} - x^{(j)}. \tag{10}$$

I use a coma in $r_{i,i+1}$ to avoid confusion but r_{ij} in general. Similarly,

$$\frac{\partial g_i}{\partial x^{(i+1)}} = -\frac{1}{\|r_{i,i+1}\|} r_{i,i+1}. \tag{11}$$

Let's introduce the unit vectors $u_{ij} = (1/\|r_{ij}\|) r_{ij}$ and so when you put all these constraints together in a matrix form, you have

$$G = \begin{bmatrix} u_{12} & -u_{12} & 0 & 0 & \dots & 0 \\ 0 & u_{23} & -u_{23} & 0 & \dots & 0 \\ 0 & 0 & u_{34} & -u_{34} & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & u_{n-2,n-1} & -u_{n-1,n} & 0 \\ 0 & 0 & 0 & 0 & u_{n-1,n} & -u_{n-1,n} \end{bmatrix}. \tag{12}$$

And that's very, very sparse.

This is all there is to distance constraints and as it turns out, except for a small issue regarding quaternions, that's the same for rigid bodies.

4 Rigid body constraints

For rigid bodies, the generalized coordinates are labeled q because they are not only Cartesian, but contain also the rotational degrees of freedom. Though there are only three of these, and though the angular velocity ω is a simple 3D vector, the rotation of the body is best represented as a quaternion. Well, let's write the velocity vector of a rigid body with coordinate x as

$$v = \begin{bmatrix} \dot{x} \\ \omega \end{bmatrix}. \tag{13}$$

That's not the same as $\dot{q}$ because if we use the position x and the quaternion e as generalized coordinates, that's a 7 dimensional vector. What happened? Well, the velocity of the quaternion is given by the well known formula

$$\dot{e} = \frac{1}{2}\omega \star e, \tag{14}$$

where ω is here a pure imaginary quaternion. I did not change the letter for it because there is a straight forward way to promote 3D vectors to quaternions. If we separate quaternions in terms of real (scalar) and imaginary components (a 3D vector), we can write

$$e = \begin{bmatrix} e_s \\ \mathbf{e}_v \end{bmatrix} \tag{15}$$

and so for the 3D vector ω, or any other vector for that matter, we have

$$\omega = \begin{bmatrix} 0 \\ \omega \end{bmatrix} \tag{16}$$

with a flagrant abuse of notation. The quaternion product in Eqn. (14) can be written in matrix form

$$\begin{aligned} \dot{e} &= \frac{1}{2}\omega \star e = \frac{1}{2}\mathcal{G}^T(e)\omega \\ \mathcal{G}(e) &= \begin{bmatrix} -\mathbf{e}_v & e_s I_3 - \widehat{e}_v \end{bmatrix}, \end{aligned} \tag{17}$$

and the convention used there is that the cross product of two 3D vectors $x \times y$ can be represented as

$$x \times y = \widehat{x}y = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}. \tag{18}$$

If you look at Eqn. (17), you can now see that it is possible to write

$$\dot{q} = K(q)v, \tag{19}$$

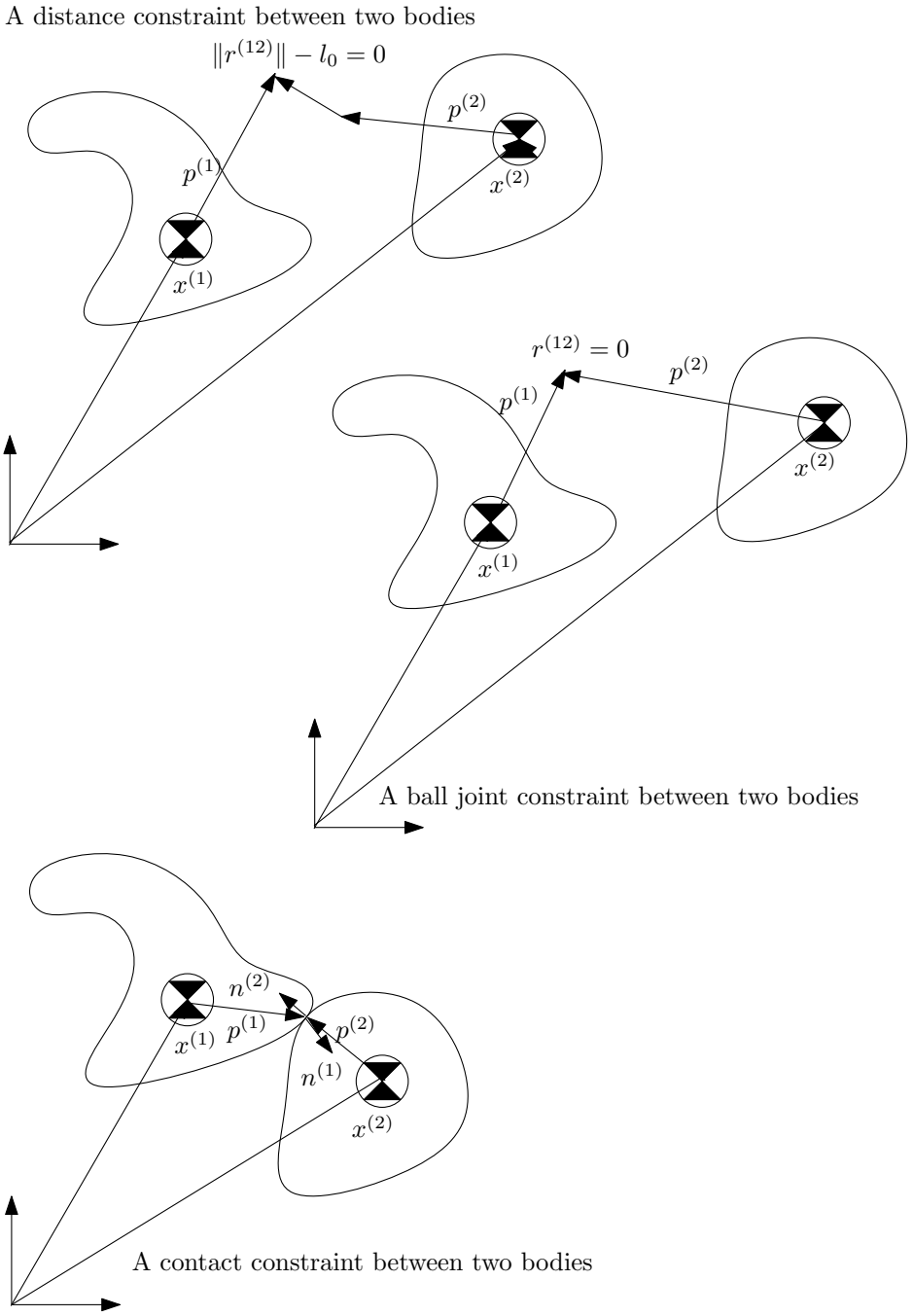

Figure 2: Various constraints between two rigid bodies which all rely on the same basic Ball Joint Jacobian. Note that normals at contacts are not unambiguously defined.

where v only contains $\dot{x}$ and ω, and $K(q)$) is a configuration dependent matrix that depends on the quaternions of your system of rigid bodies. Annoying as it may seem, that turns out to be a minor problem only.

Now, generally, if you have a constant angular velocity ω and want to integrate the differential equation in Eqn. (14) from $t = 0$ to $t = h$, you get the result

$$e(h) = \exp(\frac{h}{2}\omega)e(0), \tag{20}$$

and the exponential of a pure imaginary quaternion ω is itself a quaternion, though not pure imaginary, and it is defined as

$$\exp(\frac{h}{2}\omega) = \begin{bmatrix} \cos(\frac{h}{2}\|\omega\|) \\ \frac{\sin(\frac{h}{2}\|\omega\|)}{\|\omega\|}\omega. \end{bmatrix} \tag{21}$$

So when you integrate your multibody system, you can do all your computations for the velocity v as defined in Eqn. (13), and then update your quaternions as in Eqn. (21), which will keep unit quaternions as you integrate along. Good quaternion libraries will generate a quaternion which produces a rotation of $(h/2)\|\omega\|$ in the direction of ω without you having to do much calculation, and you can then just do a quaternion product to update.

People use different formulae for the update but that's probably the simplest one to code and understand, though not the most clever or mathematically elegant. But it's far better than doing

$$\begin{aligned} \tilde{e}_{k+1} &\longleftarrow e_k + \frac{h}{2}\mathcal{G}^T(e_k)\omega_{k+1} \\ e_{k+1} &\longleftarrow \frac{1}{\|\tilde{e}_{k+1}\|}\tilde{e}_{k+1} \end{aligned} \tag{22}$$

DON'T DO THIS!

What's the point of all this? Well, remember the stepping formula for constrained systems

$$\begin{aligned} \begin{bmatrix} M & -G^T \\ G & T \end{bmatrix} \begin{bmatrix} v_{k+1} \\ \lambda \end{bmatrix} &= \begin{bmatrix} Mv_k + hf_k \\ -\frac{\phi}{h}g_k + \phi G v_k \end{bmatrix}, \\ q_{k+1} &= q_k + hv_{k+1} \end{aligned} \tag{23}$$

where ϕ is some parameter, and T is a block diagonal positive definite matrix. SPOOK is only one specific choice of these two. Except for the last line, everything can be done using only the velocities. But now you know how to get your new rigid body configuration including the quaternions given velocities, so all you need to change is the last line in Eqn. (23).

But something else slipped in here. When we assume that $\dot{q} = v$, we have

$$g(q_{k+1}) = g(q_k + hv_{k+1}) = g(q_k) + h\frac{\partial g}{\partial q}v_{k+1}. \tag{24}$$

But that's not the case anymore. Instead, we need to look back at the definition

$$\dot{g}(q) = \frac{\partial g}{\partial q}\dot{q} = \frac{\partial g}{\partial q}K(q)v, \tag{25}$$

and that's the formula we want. Again, by abuse of notation, for the case of rigid bodies we still write

$$\dot{g} = Gv \tag{26}$$

and we call G the Jacobian. For this case, this is no longer just the gradient of g but a linear mapping between the velocity space of the rigid bodies and the constraints themselves.

What we want to do now is to avoid completely the computation of quaternion derivatives to get our precious Jacobian. How can that ever work? Consider a simple case now of a body fixed vector $\bar{p}$ which has world coordinates $p = R(e)\bar{p}$, and $R(e)$ is the orthogonal transform matrix that maps body-fixed vectors to world coordinates. The thing to remember here is that

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \Omega R = \widehat{w}R, \tag{27}$$

and since we're assuming that $\bar{p}$ is constant, then

$$\begin{aligned} \dot{p} &= (\dot{R})\bar{p} + R(\dot{\bar{p}}) \\ &= (\widehat{\omega}R)\bar{p} + R \cdot 0 = \widehat{\omega}p \\ &= \omega \times p = -p \times \omega = -\widehat{p}\omega. \end{aligned} \tag{28}$$

So, for instance, if the constraint was $g(q) = x + p$ where x is the center of mass of a body, this would pin the point $\bar{p}$ on the body at the origin. The Jacobian of that can now be computed easily from Eqn. (26) since

$$\begin{aligned} \dot{g} = \dot{x} - \widehat{p}\omega = \begin{bmatrix} I & -\widehat{p} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \omega \end{bmatrix}, \text{ and so} \\ G = G^{(\mathrm{BJ})} = \begin{bmatrix} I & -\widehat{p} \end{bmatrix}. \end{aligned} \tag{29}$$

I call that the ball-joint constraint for the good reason that if $g(q) = 0$, we have locked 3 degrees of freedom on the body and that's really just that: a ball joint. Had we used Euler angles to define our body kinematics, this ball joint would suffer *gimbal lock* for some rotation, and that's just annoying, and unnecessary. The Jacobian $G^{(\mathrm{BJ})}$ in Eqn. (29) never goes "bad", i.e., it is never row-rank deficient because the first block is the identity matrix.

We almost there for defining contact constraints. Consider two bodies and points $\bar{p}^{(1)}, \bar{p}^{(2)}$ fixed on them. The same superscript convention is used for velocities and coordinates. Now, introduce the vector

$$r = x^{(1)} + p^{(1)} - x^{(2)} - p^{(2)} \tag{30}$$

which goes from point $p^{(1)}$ to $p^{(2)}$ in world frame. The full Jacobian for this now, considering both bodies, is just as easy to derive, namely

$$\dot{r} = \begin{bmatrix} I & -\widehat{p}^{(1)} & -I & +\widehat{p}^{(2)} \end{bmatrix} \begin{bmatrix} \dot{x}^{(1)} \\ \omega^{(1)} \\ \dot{x}^{(2)} \\ \omega^{(2)} \end{bmatrix}. \tag{31}$$

So, let's now constrain the distance with $g(q) = \|r\| - l$. That's almost too easy by now

$$\begin{aligned} \dot{g} = \dot{\|r\|} &= \frac{1}{\|r\|} r^T \dot{r} \\ &\frac{1}{r} r^T G^{(\mathrm{BJ})} v. \end{aligned} \tag{32}$$

I've abused notation again by writing $G^{(\mathrm{BJ})}$ for the Jacobian of the ball joint involving two bodies, and removed all indices on v and r.

Let's assume now that we have a contact and maybe some penetration so that $r \leq 0$. Now, someone needs to give us a normal vector n along which to separate, since $r \approx= 0$, which means that it may or may not be difficult to normalize. What we'd like is that the velocities separate at least so we want

$$n^T G^{(\mathrm{BJ})} v = G^{(\mathrm{contact})} v \geq 0. \tag{33}$$

But what about tangential directions and frictional contact forces? That's easy too. Consider s, t as two orthogonal vectors, and orthogonal to n as well. This is yet more notational abuse because I already used t for time and n for the number of bodies, but don't worry. The point here is that we want to restrict the relative motion $\dot{r}$ along both direction so we want in fact $t^T \dot{r} = s^T \dot{r} = 0$ if we have stiction mode. You can just replace n in Eqn. (33) to get what you need.

That leaves the annoying issue of the mass matrices because when you do Gauss-Seidel iterations, you basically work with the Schur complement matrix $GM^{-1}G^T + T$. For rigid bodies, contrary to particles, this matrix is not just a multiple of the identity. Instead, for a single rigid body, we have

$$M = \begin{bmatrix} mI & 0 \\ 0 & \mathcal{J} \end{bmatrix} = \begin{bmatrix} mI & 0 \\ 0 & R\mathcal{J}_0 R^T \end{bmatrix} \tag{34}$$

where $\mathcal{J}_0$ is the inertia tensor in body frame, which is represented as a symmetric, positive definite 3×3 matrix, and R is the rotation matrix of the rigid bodies. The matrix I in the north-west corner is the 3×3 identity matrix. So this is now a 6×6 block matrix which needs to be updated at each step. The fact is though that for spheres and boxes, the inertia tensor is a multiple of the identity which is to say that $\mathcal{J} = \mathcal{J}_0 = \alpha I$, where $\alpha > 0$ is a scalar. That's not true for cylinders or long boxes or other shapes. But don't worry about this too much. In fact, if you do use such

big shapes, something will go very wrong with your simulations because you need to include gyroscopic forces.

If you think of continuous time and go back to Newton-Euler's laws of motion, you can write the angular momentum $L = M\omega$ and then, the second law reads

$$\frac{\mathrm{d}L}{\mathrm{d}t} = \omega \times L + M\dot{\omega} = \tau, \tag{35}$$

where τ is an external force. This is derived exactly in the same way as I did in Eqn. (28), except that now, $L = M\omega$ is not constant in body frame. In principle, you could add this with other forces on the RHS in Eqn. (23) but that won't work. Do this and you can watch your bodies cram up on the north pole of the Riemann sphere, that is to say, all go to occupy infinity. The reason is complicated. You will find all sorts of hacks for this if you read the recommended text or the SIGGRAPH proceedings. Some people say you can just integrate the angular momentum L directly with $L_{k+1} = L_k + h\tau$ and then recover ω_{k+1} from that, along with a rotation matrix. That's fallacy. The fixes are complicated to derive and not entirely trivial to implement. So you are better off leaving them out. Still, I will use the $\mathcal{J}$ in what follows without assuming that it is just a scalar multiple of the identity.

Let's get to work now, and go back to our ball and socket Jacobian. What we want here is $GM^{-1}G^T$ for one body first, so we just do it. First let me write $G^{(\mathrm{BJ})} = G$ to simplify the rest, and note that $\widehat{p}^T = -\widehat{p}$ so now

$$\begin{aligned} GM^{-1}G^T &= \begin{bmatrix} I & -\widehat{p} \end{bmatrix} \begin{bmatrix} m^{-1}I & 0 \\ 0 & \mathcal{J}^{-1} \end{bmatrix} \begin{bmatrix} I \\ \widehat{p} \end{bmatrix} = \begin{bmatrix} I & -\widehat{p} \end{bmatrix} \begin{bmatrix} m^{-1}I \\ \mathcal{J}^{-1}\widehat{p} \end{bmatrix} \\ &= m^{-1}I - \widehat{p}\mathcal{J}^{-1}\widehat{p}. \end{aligned} \tag{36}$$

The last bit looks bad in general, but remember that for distance and contact constraints, we use something like $n^T G$ instead of just G. The result of doing this with some vector $z \in \mathbb{R}^3$ for instance is

$$z^T GM^{-1}G^T n = m^{-1}z^T z + (p \times z)\mathcal{J}^{-1}(p \times z). \tag{37}$$

And that's not too bad after all.

Let's move on now to the two body case. We can partition the Jacobian as

$$G = \begin{bmatrix} G^{(1)} & G^{(2)} \end{bmatrix} \tag{38}$$

and so our computation now is

$$\begin{aligned} GM^{-1}G^T &= \begin{bmatrix} G^{(1)} & G^{(2)} \end{bmatrix} \begin{bmatrix} M^{(1)} & 0 \\ 0 & M^{(2)} \end{bmatrix} \begin{bmatrix} G^{(1)} \\ G^{(2)} \end{bmatrix} \\ &= G^{(1)}{M^{(1)}}^{-1}{G^{(1)}}^T + G^{(2)}{M^{(2)}}^{-1}{G^{(2)}}^T. \end{aligned} \tag{39}$$

In other words, we just get a simple sum of terms.

Notice that the matrix $GM^{-1}G^T$ has units of inverse mass and so $(GM^{-1}G^T)^T$ is some kind of inertia of the composite body when you apply a force of the form $f = G^T\lambda$ to it, i.e., a force that attempts to violate the constraint. But don't stretch the analogy too much because $GM^{-1}G^T \neq 0$ for a body in contact with the ground but nevertheless, it has infinite inertia against forces that push it to the ground.