Answer Sheet for

Computational Robot Dynamics 2022

Instructor: Roy Featherstone, Italian Institute of Technology
Website: http://royfeatherstone.org/teaching

Answer Mod1

```
function  Ka = kappa(lam)
  for i = 1:length(lam)
    if lam(i) == 0
      Ka{i} = i;
    else
      Ka{i} = [Ka{lam(i)} i];
    end
  end
end

function  Nu = nu(lam)
  Nu = cell(1,length(lam));
  for i = length(lam):-1:1
    Nu{i} = [i Nu{i}];
    if lam(i) ~= 0
      Nu{lam(i)} = [Nu{i} Nu{lam(i)}];
    end
  end
end
```

```
function  Mu = mu(lam)
  Mu{1} = [];
  for i = 1:length(lam)
    Mu{i+1} = [];
    Mu{lam(i)+1} = [Mu{lam(i)+1} i];
  end
end


function  d = depth(lam)
  for i = 1:length(lam)
    if lam(i) == 0
      D(i) = 1;
    else
      D(i) = D(lam(i))+1;
    end
  end
  d = max(D);
end
```

The line `Nu = cell(1,length(lam));` creates a cell array whose elements are empty arrays.

Answer Mod2

(a) A cylindrical joint allows both rotation about and translation along its axis; so it has two degrees of motion freedom (therefore two joint variables), and it can be regarded as a combination of a revolute and a prismatic joint in series.

$$\boldsymbol{X}_{\mathrm{J}}(\theta,d) = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{r}\times & \boldsymbol{E} \end{bmatrix}, \quad \boldsymbol{E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix}, \quad \boldsymbol{r}\times = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -d \\ 0 & d & 0 \end{bmatrix}, \quad \boldsymbol{S} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. The order of the columns in $\boldsymbol{S}$ is consistent with θ being the first joint variable and d the second.

(b) A helical joint also allows both rotation about and translation along its axis, but this time the translation is geared to the rotation so that the joint has only one degree of freedom overall. The translation is simply h times the rotation.

$$\boldsymbol{X}_{\mathrm{J}}(\theta) = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{r}\times & \boldsymbol{E} \end{bmatrix}, \qquad \boldsymbol{E} = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix}, \qquad \boldsymbol{r}\times = \begin{bmatrix} 0 & 0 & h\theta \\ 0 & 0 & 0 \\ -h\theta & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{S} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ h \\ 0 \end{bmatrix}$$

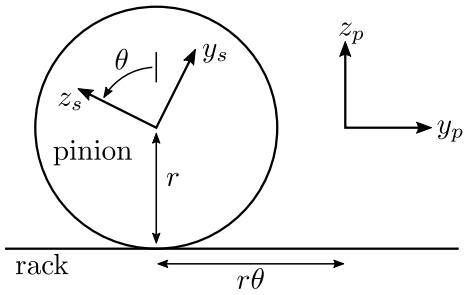
where $c = \cos(\theta)$ and $s = \sin(\theta)$.

(c) A rack-and-pinion joint also combines a rotation with a geared translation, but this time the translation is at right angles to the rotation axis. To come up with a correct joint model, one must remember that the rotation centre is not the centre of the pinion but the line of contact with the rack. As stated in the question, the successor frame has been placed in the pinion such that its x axis coincides with the central axis of the pinion. Given this placement, the z coordinate of the line of contact is $-r$ and the y coordinate is $-r\theta$, both expressed in the predecessor's coordinate system. (See diagram below.) The coordinate transform from predecessor to successor coordinates consists of a translation by $r\theta$ in the $-y$ direction followed by a rotation by θ about the x axis:

$$\boldsymbol{X}_{\mathrm{J}}(\theta) = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{r}\times & \boldsymbol{E} \end{bmatrix}, \qquad \boldsymbol{E} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix}, \qquad \boldsymbol{r}\times = \begin{bmatrix} 0 & 0 & -r\theta \\ 0 & 0 & 0 \\ r\theta & 0 & 0 \end{bmatrix},$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. The joint's motion subspace matrix is a pure rotation about the line of contact, expressed in successor coordinates. This line is always parallel to the x axis, but it moves around a circle in the y–z plane as the pinion rolls, so that it passes through the point $(0, -r\sin(\theta), -r\cos(\theta))$. The formula for $\boldsymbol{S}$ is therefore

$$\boldsymbol{S} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -r\cos(\theta) \\ r\sin(\theta) \end{bmatrix}.$$



This is an example of a joint in which $\boldsymbol{S}$ depends on the joint variable; so $\boldsymbol{S}$ is not a constant in successor coordinates and $\mathring{\boldsymbol{S}} \neq \boldsymbol{0}$, where $\mathring{\boldsymbol{S}}$ is the apparent derivative of $\boldsymbol{S}$ in successor coordinates. `spatial_v2` does not support this kind of joint directly. To improve `spatial_v2` so that it did support this kind of joint, it would be necessary to modify the function `jcalc` on Slide 12 so that it accepts one extra argument (the joint velocity variable $\dot{\boldsymbol{q}}$) and returns one more result ($\mathring{\boldsymbol{S}}\dot{\boldsymbol{q}}$).

Answer ID1

```
function  [H,C] = HCcalc(model,q,qd)
  qdd0 = 0*qd;
  C = ID(model,q,qd,qdd0);
  for i=1:model.NB
    qdd = qdd0;
    qdd(i) = 1;
    H(:,i) = ID(model,q,qd,qdd) - C;
  end
end
```

This is by far the simplest way to implement forward dynamics, given a working inverse dynamics program, but it is inefficient. A better method is to use the composite-rigid-body algorithm.

Answer Effi1

If the modification has been done correctly then the two robots should have the same inverse dynamics. The only tricky bit is making the two point masses both lie on the joint axis. In link 2 coordinates, the mass has to be positioned at the origin, whereas in link 1 coordinates it has to be positioned at point $(1, 0)$. Here is some code that will do the job:

```
m = 1;
rob1 = planar(2);
rob2 = planar(2);
rob1.I{1} = rob1.I{1} - mcI(m,[1;0],0);
rob1.I{2} = rob1.I{2} + mcI(m,[0;0],0);
```

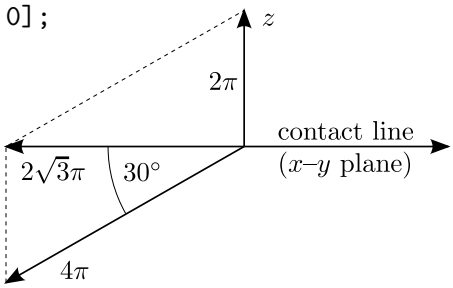
To check that the two robots have the same inverse dynamics, simply call `ID` two or three times using random values for `q`, `qd` and `qdd` and compare the results. If `tau1` and `tau2` are the results using `rob1` and `rob2` then the best way to compare them is to use an expression like `max(abs(tau1-tau2))`, which will tell you the largest absolute difference between them. If this is a number around 10^{-15} then the difference is due to rounding error and can be ignored.

Answer FD1

$\lambda = [0, 1, 1, 3, 3]$. Yes, A does have the same sparsity pattern as L, and L2 is identical to L. L3 is also identical to L, and the point here is that the sparse factorization algorithms expect certain elements of the matrix to be zero, but do not require the remaining elements to be nonzero. Given $\lambda = [0, 1, 2, 3, 4]$, which is the parent array for an unbranched tree, `LTL` does not expect any elements to be zero, and therefore operates on the whole matrix (i.e., it performs a non-sparse factorization).

Answer Sim1

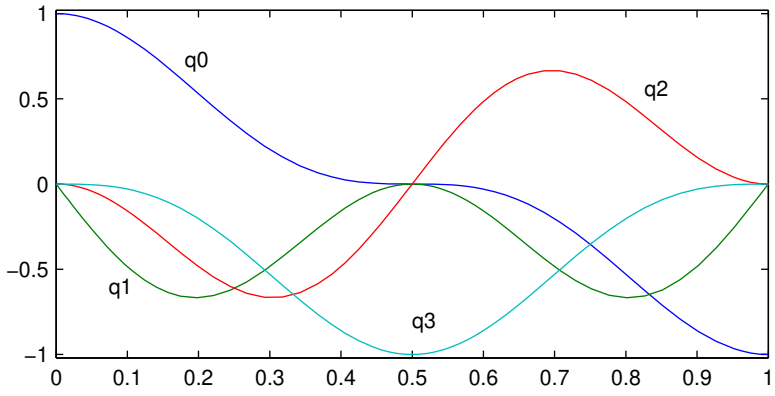
```
function  w = conew( t )
  w = 2*sqrt(3)*pi * [-cos(2*pi*t); -sin(2*pi*t); 0];
end

function  qd = coneqd( t, q )
  w = conew(t);
  qd = rqd(w,q);
end
```



We are told that the cone rolls without slipping, so the instantaneous rotation axis must be the line of contact. We are also told that the cone revolves once per second around the z axis, starting on the x axis at $t = 0$, so the unit vector on the line of contact must be $[\cos(2\pi t)\ \sin(2\pi t)\ 0]^{\mathrm{T}}$. So all that remains is to calculate the magnitude. Referring to the diagram above, the cone can be regarded as rotating about the z axis with an angular velocity of 2π, while simultaneously rotating about its own axis with an angular velocity of -4π. This axis is inclined at 30° to the line of contact, and therefore points up and to the right; but the velocity is drawn in the opposite direction because of its negative magnitude. The sum of these two vectors is

$$\boldsymbol{\omega} = -2\sqrt{3}\pi \begin{bmatrix} \cos(2\pi t) \\ \sin(2\pi t) \\ 0 \end{bmatrix}.$$

In `coneqd` the only tricky bit is to realise that $\boldsymbol{\omega}$ is expressed in the fixed coordinate frame, and therefore must be the first argument to `rqd`.



The plot of $q(t)$ should look like this. The cone ends at the same orientation at which it began, although the quaternion has changed sign. Half way through, the cone has made one complete rotation about its own axis, but only half a rotation about the z axis, and the quaternion has the value $[0, 0, 0, 1]$, which represents a rotation of π about the z axis.

Answer Sim2

```
function  v = conev( t )
   w = conew(t);
   pos = [0;1;0];
   v = [ w; cross(pos,w) ];
end
```

```
function  pd = conepd( t, p )
   q = p(1:4);
   r = p(5:7);
   v = conev(t);
   w = v(1:3);
   vo = v(4:6);
   qd = rqd(w,q);
   rd = vo - cross(r,w);
   pd = [qd;rd];
end
```

Both of these functions make use of the velocity shift formula

$$\boldsymbol{v}_O = \boldsymbol{v}_P + \overrightarrow{OP} \times \boldsymbol{\omega}\,,$$

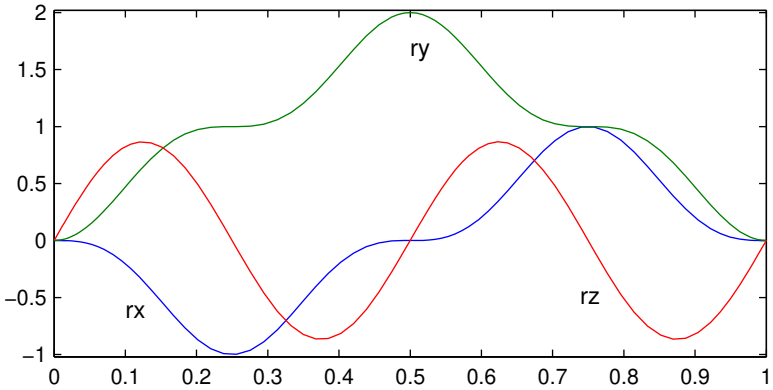
which expresses the relationship between the velocity of the body-fixed point at O and the velocity of the body-fixed point at P. To get the correct answer for `conev`, we place P at the apex of the cone, so that $\boldsymbol{v}_P = \mathbf{0}$ (because the rotation axis always passes through this point), which lets us calculate $\boldsymbol{v}_O$ as follows:

$$\boldsymbol{v}_O = \overrightarrow{OP} \times \boldsymbol{\omega}\,, \qquad \text{where } \overrightarrow{OP} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

In `conepd` the tricky part is to realize that $\boldsymbol{r}$ is tracking the motion of a body-fixed point, and so $\dot{\boldsymbol{r}}$ must be the velocity of that point, but the linear part of the spatial velocity is the velocity of the body-fixed point at the origin. So, in this case, P is wherever $\boldsymbol{r}$ is pointing (which is not the apex of the cone), and the objective is to calculate $\boldsymbol{v}_P$ given $\boldsymbol{v}_O$. So `conepd` calculates

$$\dot{\boldsymbol{r}} = \boldsymbol{v}_P = \boldsymbol{v}_O - \overrightarrow{OP} \times \boldsymbol{\omega}\,, \qquad \text{where } \overrightarrow{OP} = \boldsymbol{r}\,.$$

In part (c) the plot of the quaternion part of $\boldsymbol{p}$ should be the same as the plot in question Sim1(c). The plot of the positional part should look like this:



The position begins and ends at $(0, 0, 0)$. Half way through, it has reached the position $(0, 2, 0)$. This is because the cone has made one complete revolution about its axis, but only half a revolution about the vertical line through $(0, 1, 0)$, so the body-fixed point originally at the origin is now on the other side of $(0, 1, 0)$. At the beginning and end of the motion, the point is moving straight up. At $t = 0.25$, the cone has made a half revolution about its axis and a quarter revolution about the vertical line, so the body-fixed point is back in the x–y plane at coordinates $(-1, 1, 0)$. A similar situation arises at $t = 0.75$.

In part (d), you should get no motion of $\boldsymbol{r}$ because it is tracking the body-fixed point at the cone's apex.

Answer Sim3

```
function  a = conea( t )
  wd = 4*sqrt(3)*pi^2 * [sin(2*pi*t); -cos(2*pi*t); 0];
  pos = [0;1;0];
  a = [ wd; cross(pos,wd) ];
end
```

As the spatial velocity is expressed in a stationary coordinate system, the spatial acceleration is simply the time derivative of the velocity:

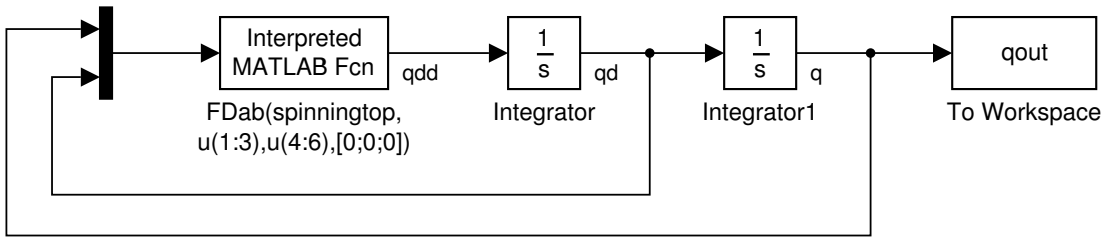
$$\dot{\boldsymbol{v}} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{\omega}\\ \overrightarrow{OP}\times\boldsymbol{\omega}\end{bmatrix} = \begin{bmatrix}\dot{\boldsymbol{\omega}}\\ \overrightarrow{OP}\times\dot{\boldsymbol{\omega}}\end{bmatrix}$$

where $\overrightarrow{OP}$ gives the position of the cone's apex, and is therefore a constant. If you have implemented `conea` correctly then its integral should indeed be equal to the velocity as calculated by `conev`, except for truncation error in the integration process.

Answer Sim4

There are several possible answers to this question. One possible model function and one possible Simulink model are shown below.

```
function  top = spinningtop

  persistent memory;                    % for efficient use with Simulink

  if length(memory) > 0                 % robot model already exists
    top = memory;
    return
  end

  R = 0.05;                             % radius of top
  h = 0.1;                              % height of CoM above pivot
  m = 1;                                % mass of top
  % spatial inertia of top: thin disc of radius R
  Itop = mcI( m, [0,0,h], m*R^2*diag([1/4,1/4,1/2]) );

  top.NB = 3;
  top.parent = [0 1 2];
  top.jtype = { 'Rz', 'Ry', 'Rz' };
  top.Xtree = { eye(6), eye(6), eye(6) };
  top.I = { zeros(6), zeros(6), Itop };

  top.appearance.base = { 'tiles', [-0.1 0.1; -0.1 0.1; 0 0], 0.05 };
  top.appearance.body{3} = {...
    'facets', 12,...
    'cyl', [0 0 h-0.005; 0 0 h+0.005], R,...
    'cyl', [0 0 0; 0 0 h+0.05], 0.0025 };

  memory = top;                         % remember for subsequent calls
end
```



The initial position was set to `[0;0.5;0]` and the initial velocity to `[0;0;1000]`. Clearly, other initial values will also work. The resulting motion is not smooth: the axis of the top wobbles as the top precesses. This is not an error or an inaccuracy. The wobble is a real part of precession motion.