Appendix C

DAP_JC User Manual

(September 2018)

The Dynamic Analysis Program: Joint Coordinates (DAP_JC) is a semi-general-purpose program that is developed to assist a user in learning how the joint-coordinate equations of motion are formulated. The program has many similarities and also many differences with the Dynamic Analysis Program: Body Coordinates (DAP_BC). The similarities are in the structure of the two programs, in the input data, and in the post-processing and the animation of the results. The main difference is that DAP_JC does not automatically construct the equations of motion—these equations for a multibody model must be provided by the user.

In this appendix we first discuss the structure of the program. Then we show how to use the program to simulate any of the models that are provided with the program. Finally we discuss in detail how to construct a model.

Most of the M-files that form the program DAP_JC have names similar to the M-files in DAP_BC however, they are not identical. The M-files from the two programs that share the same name perform the same task but with different formulations.

C.1 STRUCTURE OF DAP_JC

The program follows the formulation of the equations of motion as presented in Chapter 9 of the textbook. A model could be a system without any constraints (for example an open-chain system without a driver constraint), or with constraints (for example closed-chain system with or without a driver constraint).

For an open chain system with no added constraint such as a driver, the program considers the equations of motion from Eqs. (9.22)-(9.24):

$$\boldsymbol{M}\ddot{\boldsymbol{\theta}} = {}^{(a)}\boldsymbol{h} \tag{9.22}$$

where

$$\boldsymbol{M} = \mathbf{B}'\mathbf{M}\mathbf{B} \tag{9.23}$$

$${}^{(a)}\boldsymbol{h} = \mathbf{B}'({}^{(a)}\mathbf{h} - \mathbf{M}\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}) \tag{9.24}$$

Based on the model data provided by the user, the program constructs and evaluates the mass matrix **M** and the array of applied forces ${}^{(a)}\mathbf{h}$ containing the forces of gravity, springs, dampers, etc. The user must provide complete description of forward kinematics, the matrix **B**, and the array $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$. The program constructs matrix $\boldsymbol{M}$, array ${}^{(a)}\boldsymbol{h}$, and performs integration of the equations of motion.

For a closed chain system, or any system containing constraints, the program constructs the equations of motion according to Eq. (9.34):

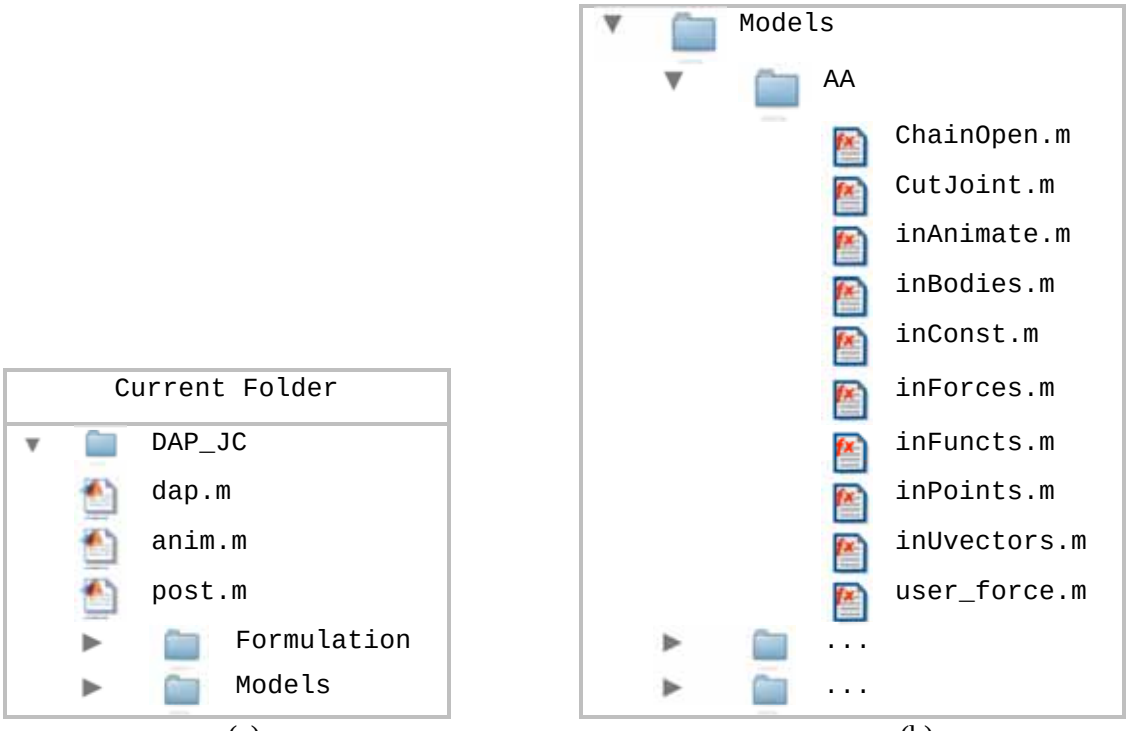
$$\begin{bmatrix} \boldsymbol{M} & -\mathbf{C}' \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \ddot{\boldsymbol{\theta}} \\ {}^{*}\boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} {}^{(a)}\boldsymbol{h} \\ -\dot{\mathbf{C}}\dot{\boldsymbol{\theta}} \end{Bmatrix} \tag{9.34}$$

where **C** is the Jacobian of the constraints (refer to Eq. (9.28)). In addition to the complete description of forward kinematics, the matrix **B** and the array $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$, the user must also supply a program to construct and evaluate the constraints, matrix **C**, and the array $\dot{\mathbf{C}}\dot{\boldsymbol{\theta}}$. Then the program constructs the equations of motion as in Eq. (9.34) and integrates the equations as discussed in Chapter 13.

In Chapter 9 we discussed that the rotational joint coordinate associated with a revolute (pin) joint could be defined either as a relative coordinate or an absolute coordinate. In the program `DAP_JC` we use **relative** joint coordinates. Therefore, if we decide to model some of the examples from Chapter 9 that are formulated based on the absolute joint coordinates, we must first reformulate them with relative joint coordinates before constructing the necessary M-files for the program `DAP_JC`.

C.2 USING DAP_JC

The program `DAP_JC` and the corresponding example models can be found in the folder named `DAP_JC` followed by the date of its last revision. As shown in Figure C.1(a), similar to `DAP_BC`, this folder contains three script M-files and two sub-folders. The folder named `Models` contains several examples of simple multibody systems, as shown in Figure C.1(b). Some of these models are discussed in detail in this manual as a guide in learning how to use the program, and some models are discussed briefly. If we construct a new model, we should save it in this folder.



(a) (b)

Figure C.1 Folders and M-files files of `DAP_JC`.

The script M-file `dap` is the main program to execute. The program prompts the user with a few questions prior to performing a simulation. After a simulation is completed, the user may execute the script M-file `anim` to visualize a crude animation of the simulated response. Or, following a simulation,

we may execute the script M-file `post` for post-processing of the results. This script transforms the results into a more recognizable set of arrays for further analyses or plots.

The folder `Formulation` contains other subfolders, script and function M-files. To simulate the response of a multibody system, using the existing capabilities of the program, there is no need for a user to revise any of the M-files in this folder.

It is highly recommended to make the folder `DAP_JC` the "Current Folder" in MATLAB, as indicated in Figure C.1. This facilitates setting the "paths" to different folders and M-files.

Executing the programs in `DAP_JC` is very similar to that in the programs in `DAP_BC`. In the Command Window we type `dap` and then respond to each question:

```
>> dap
Which folder contains the model?
```

If we enter the name of a folder that contains an open-chain model, the program will continue with the next question for the final time and time step. But if the model is a closed-chain system, the program will ask if we want to correct the initial conditions on the "joint" coordinates and velocities:

```
Do you want to correct the initial conditions? [(y)es/(n)o]
```

If we are certain that the provided initial conditions satisfy the position and velocity constraints of the cut joint(s) or any other existing constraints, we may answer no, otherwise we should respond yes. If we respond yes, the program will report the corrected values for the joint coordinates and velocities. Next, the program will ask for the final time of integration and the reporting time steps (all in seconds):

```
Final time = ?
Reporting time-step = ?
```

During the integration process for the requested time period, the program keeps the user informed of the progress by reporting the integration time every 100 function-evaluations. (A function evaluation means that the equations of motion have been constructed and solved for the accelerations.) At the completion of integration, the program reports the number of function evaluations. This information is reported as a measure for the efficiency (or inefficiency) of the method of solution.

After the completion of a simulation, we can observe an animation of the results by typing in the Command Window `anim`, or executing `post` for further post-processing of the results:

```
>> anim
>> post
```

Some of the existing models in the `Models` folder are:

`AA`:	Double A-arm suspension (closed-chain)
`Cart_C`:	Cart (closed-chain)
`MPA`:	MacPherson suspension with three moving bodies (closed-chain)
`MPB`:	MacPherson suspension with two moving bodies (closed-chain)
`Rod`:	A single rod falling and impacting the ground (open-chain)
`Rod2`:	Two connected rods falling and impacting the ground (open-chain)
`SC`:	Slider-crank (closed-chain)
`SCd`:	Slider-crank with driver motor (closed-chain)
`TP`:	Triple pendulum (open-chain)

C.3 CONSTRUCTING A MODEL

Most of the function M-files for a model are exactly the same as those for `DAP_BC`. As an example, the input files for the double A-arm system `AA` are shown in Figure C.1(b) where most of the file names should be familiar. The model input files for both programs are listed next to each other in the following

table for comparison. The files that are marked with an asterisk are exactly the same in both programs. The M-file `inBodies` for `DAP_JC` contains slightly different data as in `DAP_BC`, which will be discussed later in this manual. The M-file `inJoints` that is required in `DAP_BC` is not needed for `DAP_JC`, but instead we have a file named `inConst` in `DAP_JC`.

There are two additional files that a model in `DAP_JC` may require that contain formulations—not only data like the other input files. The M-file `ChainOpen`, that is required for every model, must describe the forward kinematics of an open-chain or a cut open-chain system. The M-file `CutJoint` is only required for a closed-chain system describing the cut-joint and/or if we have any driver constraints in a system. This file is not needed if the system is open-chain and there are no driver constraints.

DAP_JC	DAP_BC	Description
inAnimate *	inanimate *	Animation data
inBodies	inBodiess	Body data
inConst		Stating the number of constraints if any
inForces *	inForces *	Applied force data
inFuncts *	inFuncts *	Time function type and data
	inJoints	Kinematic joint data
inPoints *	inPoints *	Point data
inUvectors *	inUvectors *	Unit vector data
user_force *	user_force *	Non-standard user force model
ChainOpen		Open chain kinematics
CutJoint		Cut joint(s) constraint entities
* These files are the same in both programs. For a description of these files refer to the user manual for DAP_BC.		

To construct a model, we should follow practically the same steps as in `DAP_BC` for indexing bodies, points, and unit vectors. The main difference here is that for an open-chain system, whether it is cut or originally open, we **must** index the bodies in an **ascending** order starting from the ground (as body "0") and continuing towards the leaves.

M-file `inBodies`: In this file we define the mass and moment of inertia for the bodies, and initial conditions for all the joint coordinates and velocities. If a body is the owner of a floating joint, we must define three joint coordinates—the translational and rotational coordinates of the body—and the corresponding joint velocities. If the body is the owner of a revolute or a translational joint, then only one joint coordinate and one joint velocity needs to be defined.

```
function inBodies
       include_global

   B1 = Body_struct;
   B1.m = 5.0; B1.J = 4.0;
   B1.theta = [1.0 0.5 pi/6]; % floating joint
   B1.theta_d = [0 0 0];   % default values

   B2 = Body_struct;
   B2.m =3.0; B1.J = 2.0;
   B2.theta = 0.5;   % revolute or translational joint
   B1.theta_d = 0;   % default value

   ...
```

```
   Bodies = [B1; B2; ...];
```

M-file `inPoints`: In this file we provide body-fixed coordinates, $\mathbf{s}_i^P$, for all the defined points, including the points on the ground. Construction of this file is the same as that of `DAP_BC`.

M-file `inUvectors`: In this file we provide the body-fixed components, $\mathbf{u}_i$, of the defined unit vectors, including the unit vectors on the ground. Construction of this file is the same as that of `DAP_BC`.

M-file `inForces`: In this file we define the forces that act on the bodies, such as gravity, spring-dampers, or other force elements. Construction of this file is the same as that of `DAP_BC`.

M-file `inFuncts`: A user may define a function such as $f = f(t)$, and apply it to describe the kinematics of a driver motor, for example, as function of time. The structure for these elements is the same as that of `DAP_BC`.

M-file `inAnimate`: This file provides data for the animation of the simulated response. Construction of this file is the same as that of `DAP_BC`.

M-file `user_force`: In this file we define nonstandard forces/torques that act on the bodies. Construction of this file is the same as that of `DAP_BC`.

There is no need to construct an M-file `inJoints` as needed in `DAP_BC`. The necessary cut-joint constraints are formulated in the M-file `CutJoint` as will be described in the upcoming sections.

M-file `inConst`: In this file we state the number of constraints that exists in a model either due to the cut joints or drivers. The number of constraints associated with different cut joints and driver is listed in Table C.1.

Table C.1 Cut joints and the corresponding number of constraints

Constraint type		**Number of constraints**
Cut joint	Revolute (pin)	2
	Translational (sliding)	2
	Revolute-revolute	1
	Revolute-translational	1
Driver		1

```
function inConst
    include_global

    nConst = 2;
```

If there are no constraints in a model, we still need to provide this M-file and state that `nConst = 0`.

M-file `ChainOpen`: The user must provide this M-file for any model, whether an open-chain or a cut open-chain system. The M-file contains three parts: (a) describing the forward kinematics of the system; (b) constructing the **B** matrix; and (c) constructing the $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$ array.

```
function ChainOpen(check, theta, theta_d, t)
% Recursive coordinate transformations
    include_global

if check <= 2
    (a) Provide statements for forward kinematics to update coordinates
        of points and components of vectors
end
if check == 2
    (b) Provide statements to construct and evaluate matrix B
elseif check == 3
    (c) Provide statements to construct and evaluate Bθ̇θ̇ array
end
end
```

In this function M-file the value of the parameter `check` directs the function to evaluate the required entities. This parameter is set by the parent function to 1, 2 or 3. The following is a more detailed description of the three required statements.

(a) *Forward kinematics* (`check = 1 or 2`)

In this part the user must provide statements to compute the coordinates of every point in the system starting from the base and moving toward every leaf. This computation requires updating:

$\mathbf{A}_i$ for all the bodies

$\mathbf{s}_i^P$ and $\mathbf{r}_i^P$ for all the points

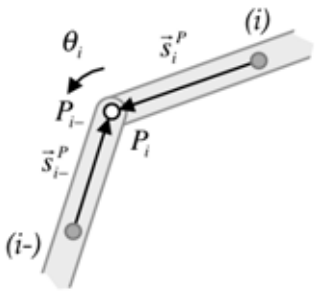
$\mathbf{u}_i$ for all the unit vectors

For these computations the user should follow the recursive kinematic process that is described in Section 9.1.2. To simplify this process for the user, the program provides the following three functions based on the type of kinematic joint that exists in the open-chain path.

Revolute Joint

```
function Coord_Rev(Pim, Pi, theta)
```

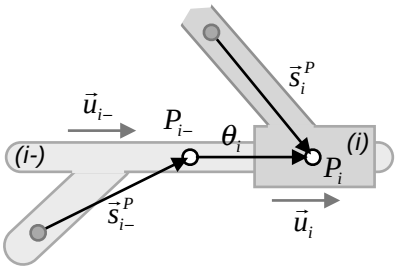
- The user must provide to this function indices of points P_{i-} and P_i, and the value of the joint coordinate θ_i.
- This function assumes that the rotational transformation matrix $\mathbf{A}_{i-}$ has already been evaluated. The function evaluates $\mathbf{s}_{i-}^P$, $\bar{\mathbf{s}}_{i-}^P$, $\mathbf{r}_{i-}^P$, ϕ_i, $\mathbf{A}_i$, $\mathbf{s}_i^P$, $\bar{\mathbf{s}}_i^P$, $\mathbf{r}_i^P$, and $\mathbf{r}_i$.



Translational Joint

```
function Coord_Tran(Pim, Uim, Pi, Ui, theta)
```

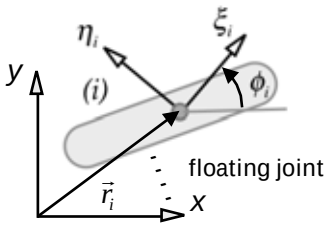
- The user must provide to this function indices of points and unit vectors P_{i-}, $\vec{u}_{i-}$, P_i and $\vec{u}_i$, and the value of the joint coordinate θ_i.
- This function assumes that the rotational transformation



matrix $\mathbf{A}_{i-}$ has already been evaluated. The function evaluates $\mathbf{s}_{i-}^P$, $\breve{\mathbf{s}}_{i-}^P$, $\mathbf{r}_{i-}^P$, $\mathbf{u}_{i-}$, $\breve{\mathbf{u}}_{i-}$, ϕ_i, $\mathbf{A}_i$, $\mathbf{s}_i^P$, $\breve{\mathbf{s}}_i^P$, $\mathbf{r}_i^P$, $\mathbf{u}_i$, $\breve{\mathbf{u}}_i$ and $\mathbf{r}_i$.

Floating Joint

```
function Coord_Float(Bi, theta)
```



- The user must provide to this function the index of the body, and the value of three joint coordinates.
- This function assigns the joint coordinates to ϕ_i and $\mathbf{r}_i$, then it evaluates matrix $\mathbf{A}_i$.

In addition to the points and vectors in the open chain path that are updated, any other points and vectors that are not in the path (for example those that belong to the force elements), their coordinates and components must also be updated. To simplify this process for the user, the program provides the following two functions.

Update Point

```
function Update_P(Pi)
```

- The user must provide to this function the index of the point that its coordinates need to be updated.
- This function updates $\mathbf{s}_i^P$, $\breve{\mathbf{s}}_i^P$, and $\mathbf{r}_i^P$.

Update Vector

```
function Update_U(Ui)
```

- The user must provide to this function the index of the vector that its components need to be updated.
- This function updates $\mathbf{u}_i$, and $\breve{\mathbf{u}}_i$.

(b) **B** *matrix* (`check = 2`)

In this part the user must provide statements to compute the **B** matrix. At this point all the coordinates and vectors $\mathbf{s}_i^P$, $\breve{\mathbf{s}}_i^P$, $\mathbf{r}_i^P$, $\mathbf{u}_i$ and $\breve{\mathbf{u}}_i$ have been updated. We need to define and compute the components of any $\mathbf{d}_{i,j}$ vector as needed.

(c) $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$ *array* (`check = 3`)

In this part the user must provide statements to compute the $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$ array. At this point all the coordinates and velocities have been updated. We need to compute components of any $\dot{\mathbf{d}}_{i,j}$ vector as needed.

As a simple example for an open chain system for modeling with DAP_JC, we consider the sliding pendulum from Chapter 8 that we used to demonstrate how to set up a model for DAP_BC. Originally this example was considered for FBD construction in Section 6.1.3.

Example C.1 (open-chain)

To begin constructing a model for the sliding pendulum, we follow the process that is described in detail in Chapter 9. We first separate the bodies, assign indices to the bodies, attach to each a body-fixed frame, and define a global *x*-y frame. As shown in Figure C.2, the two moving bodies are marked as (*1*) and (*2*) in an ascending order that is essential for the program DAP_JC. We have identified three points as O_0, B_1 and B_2 that are needed for the joints and the spring attachment points. We have assigned indices to these points as [1], [2], and [3] (the order of numbering is up to us). We also need two unit vectors for the translational joint: $\vec{u}_0$ on the ground and $\vec{u}_1$ on the slider. These unit vectors are numbered [[1]] and [[2]]. We are now ready to set up the M-files for this model. The constant dimensions are $a = 0.2$ m and $L = 1.0$ m.

The constructed M-files for this example can be found in Models/SP.

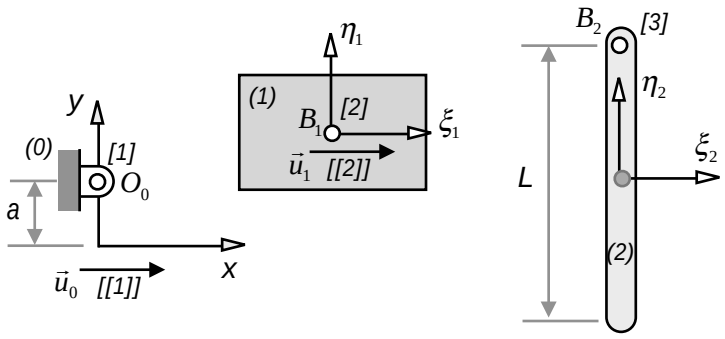


Figure C.2 Individual bodies and the assigned indices for the sliding pendulum example.

The data, for the mass, moment of inertia, and initial conditions on the joint coordinates and velocity for each body is provided in the function M-file inBodies.

$\theta_1 = 1.0$
$\dot{\theta}_1 = 0$

$\theta_2 = \pi / 6$
$\dot{\theta}_2 = 0.3\pi$ rad/sec

```
function inBodies
    include_global

B1 = Body_struct;
B1.m = 5.0; B1.J = 4.0;
B1.theta = 1.0;

B2 = Body_struct;
B2.m = 2.0; B2.J = 0.2;
B2.theta = pi/6;
B2.theta_d = 0.3*pi;

Bodies = [B1; B2];
```

The following function M-files are the same as in the DAP_BC model:
inAnimate, inForces, inFuncts, inPoints, inUvectors

Since there are no constraints in this model, we need to provide nConst = 0 in this function M-file.

```
function inConst
    include_global

    nConst = 0;
```

The last function M-file to construct is the following:

(a) *Forward kinematics*
The forward kinematics starts from the ground

```
function ChainOpen(check, theta,
                              theta_d, t)
```

and goes through the translational joint using the translation joint function `Coord_Tran`. The indices refer point O_0, unit vector $\vec{u}_0$, point B_1, and unit vector $\vec{u}_1$ respectively, and the last argument contains the value of θ_1.

Next the path goes through the revolute joint that uses the revolute joint function `Coord_Rev`. The indices refer to points B_1 and B_2, and the value of θ_2.

(b) **B** *matrix*

The **B** matrix for this example is determined as

$$\mathbf{B} = \left[\begin{array}{c|c} \mathbf{u}_0 & \mathbf{0} \\ 0 & 0 \\ \hline \mathbf{u}_0 & \breve{\mathbf{d}}_{2,2} \\ 0 & 1 \end{array}\right]$$

(c) $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$ *array* (`check = 3`)

For the sliding pendulum example, the $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$ array is determined as

$$\dot{\mathbf{B}}\dot{\boldsymbol{\theta}} = \left\{\begin{array}{c} \mathbf{0} \\ 0 \\ \breve{\mathbf{d}}_{2,2}\dot{\theta}_2 \\ 0 \end{array}\right\}$$

```
    include_global

if check <= 2

    Coord_Tran(1, 1, 2, 2, theta(1));


    Coord_Rev(2, 3, theta(2));
end


if check == 2
    u1 = Uvectors(2).u;
    d22 = -Points(3).sP;
    z2 = [0; 0];
    Bmat = [u1  z2
            0   0
            u1  s_rot(d22)
            0   1];



 elseif check == 3
    d22d = -Points(3).sP_d;

    Bd = [0
          0
          0
          s_rot(d22d)*theta_d(2)
          0];
end
end
```

Now we have all the files to simulate the response of this system.

For systems containing closed chains or drivers, we construct all the necessary M-files for the cut system, as for any other open chain system. Then we provide the following function M-file to describe the necessary entities for the cut joint and/or driver constraints.

M-file `CutJoint`: The user must provide this M-file for any model that contains cut joints and/or driver constraints. The M-file contains four parts defining: (a) the Jacobian matrix **C**; (b) the coordinate constraints; (c) the right-hand-side array of the velocity constraints; and (d) the right-hand-side array of the acceleration constraints; i.e., $-\dot{\mathbf{C}}\dot{\boldsymbol{\theta}}$.

```
function CutJoint(check, theta, theta_d, t)
% Cut joint and driver constraints
    include_global

    (a) Provide the Jacobian matrix of the constraints
if check == 1
    (b) Provide the coordinate constraints
end
if check == 2
    (c) Provide the right-hand-side array of the velocity constraints
```

```
elseif check == 3
    (d) Provide the right-hand-side array of the acceleration
        constraints
end
end
```

In this function M-file the value of the parameter `check` directs the function to evaluate the required entities. This value, which could be 1, 2 or 3, is set by the parent function. The following is a more detailed description of the four required set of statements.

(a) *Jacobian matrix* **C**
In this part the user must provide statements to compute the Jacobian matrix **C** for all the constraints. This matrix will be evaluated whether the parameter `check` is equal to 1, 2 or 3.

(b) *Cut-joint coordinate constraints*
The user must provide statements to evaluate all the constraints, whether cut joint or driver. If the coordinate values do not satisfy these constraints, the program will correct them; e.g., in the initial condition correction step. These constraints will be evaluated when the parameter `check` is set to 1 by the program.

(c) *Right-hand side array of velocity constraints*
The right-hand side array of velocity constraints for any cut joint should be zero, but for a driver constraint it could be a constant non-zero value or a time dependent function. This array will be evaluated when the parameter `check` is set to 2 by the program.

(d) *Right-hand side array of acceleration constraints* $-\dot{\mathbf{C}}\dot{\boldsymbol{\theta}}$
The right-hand side of acceleration constraints for a cut joint most likely contains quadratic velocity terms, but for a driver constraint it could be zero, constant non-zero value, or a time dependent function. This array will be evaluated when the parameter `check` is set to 3 by the program.

Next example shows how to incorporate constraints for a cut joint.

Example C.2 (closed-chain)

In this example we consider the slider-crank mechanism from Chapter 9, Example 9.4. In this model, however, we use relative angle for the rotational joint coordinate at pin joint *A* and not an absolute angle as used in Example 9.4.

To begin constructing a model for the slider-crank mechanism, we cut the system at revolute joint *B*, we separate the bodies, assign indices to the bodies, attach to each a body-fixed frame, and define a global *x*-y frame. We have identified six points as O_0, O_1, A_1, A_2, B_2 and B_3, and two unit vectors $\vec{u}_0$ and $\vec{u}_3$. We then assign indices to the points and the unit vectors as shown on the figure.

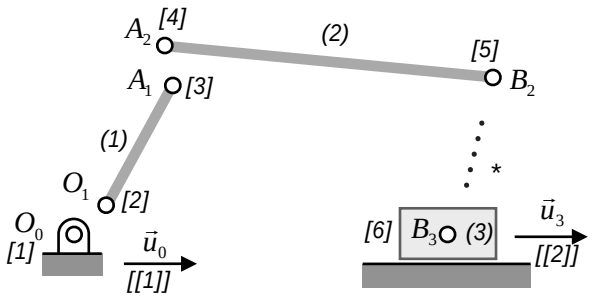


Figure C.3 The slider-crank mechanism from Example 9.4.

The constructed M-files for this example can be found in `Models/SC`. Most of the input M-files are the same as the model for the `DAP_BC` program. In the function M-file `inBodies`, for the defined three bodies, the initial conditions are stated based on the joint coordinates as $\theta_1 = \phi_1$, $\theta_2 = \phi_2 - \phi_1$, and $\theta_3 = x_3$. However, when we execute the program, we should ask the program to correct the initial conditions in case the stated values do not satisfy the cut-joint constraints.

Since we have cut a revolute joint there are two constraints in this model.

```
function inConst
    include_global

    nConst = 2;
```

The function M-file `ChainOpen` is constructed as in the following.

(a) *Forward kinematics*

Since there are two trees in this system, the forward kinematics for the first tree starts from the ground through the revolute joint O (O_0 and O_1) to body (*1*), then through the revolute joint A (A_1 and A_2) to body (*2*). For the second tree, from the ground we go through the translational joint (point O_0, unit vector $\vec{u}_0$, point B_3, and unit vector $\vec{u}_3$) to body (*3*).

Point B_2 on body (*2*) is not in the path of the forward kinematics. Therefore we update its coordinates using the `Update_P` function.

(b) **B** *matrix*

The **B** matrix for this example is determined as

$$\mathbf{B} = \left[\begin{array}{c|c|c} \breve{\mathbf{d}}_{1,1} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 0 \\ \hline \breve{\mathbf{d}}_{2,1} & \breve{\mathbf{d}}_{2,2} & \mathbf{0} \\ 1 & 1 & 0 \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{u}_3 \\ 0 & 0 & 0 \end{array}\right]$$

```
function ChainOpen(check, theta,theta_d, t)
    include_global

if check <= 2

    Coord_Rev(1, 2, theta(1));
    Coord_Rev(3, 4, theta(2));
    Coord_Tran(1, 1, 6, 2, theta(3));









    Update_P(5);

end

if check == 2
    d11 = -Points(2).sP;
    d21 =  Bodies(2).r;
    d22 = -Points(4).sP;
    u3 = Uvectors(2).u;
    z2 = [0; 0];
    Bmat = [s_rot(d11)  z2          z2
            1           0           0
            s_rot(d21)  s_rot(d22)  z2
            1           1           0
            z2          z2          u3
            0           0           0];
```

(c) $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$ *array* (`check = 3`)

The $\dot{\mathbf{B}}\dot{\boldsymbol{\theta}}$ array is constructed as

$$\dot{\mathbf{B}}\dot{\boldsymbol{\theta}} = \begin{Bmatrix} \breve{\mathbf{d}}_{1,1}\dot{\theta}_1 \\ 0 \\ \hline \breve{\mathbf{d}}_{2,1}\dot{\theta}_1 + \breve{\mathbf{d}}_{2,2}\dot{\theta}_2 \\ 0 \\ \hline \mathbf{0} \\ 0 \end{Bmatrix}$$

```
 elseif check == 3
    d11d = -Points(2).sP_d;
    d21d =  Bodies(2).r_d;
    d22d = -Points(4).sP_d;
    Bd = [s_rot(d11d)*theta_d(1)
          0
          s_rot(d21d*theta_d(1) +
                    d22d*theta_d(2))
          0
          0
          0
          0];
end
end
```

Since this model contains a cut joint, we need to provide the function M-file `CutJoint` as shown in the following. In this file we provide statements for (a) the Jacobian matrix, (b) the constraints on the coordinates, (c) the right-hand side array of the velocity constraints, and (d) the right-hand side array of the acceleration constraints.

(a) *Jacobian matrix* **C**

The **C** matrix for this example is constructed as

$$\mathbf{C} = \begin{bmatrix} \breve{\mathbf{d}}_{*,1} & \breve{\mathbf{d}}_{*,2} & -\mathbf{u}_3 \end{bmatrix}$$

Note that this matrix is different from the one in Example 9.4 because we are using relative joint coordinate for θ_2.

(b) *Coordinate constraint*

The only constraints are from the cut-joint at the pin joint *B*.

(c) *Right-hand side array of velocity constraints*

For a cut joint this is a zero array.

(d) *Right-hand side array of acceleration constraints*

For the cut pin joint this array is constructed as

$$-\dot{\mathbf{C}}\dot{\boldsymbol{\theta}} = -(\breve{\mathbf{d}}_{*,1}\dot{\theta}_1 + \breve{\mathbf{d}}_{*,2}\dot{\theta}_2)$$

```
function CutJoint(check, theta, theta_d, t)
    include_global

    dcut1 = Points(5).rP;
    dcut2 = Points(5).rP - Points(4).rP;
    u3 = Uvectors(2).u;
    C = [s_rot(dcut1)  s_rot(dcut2)   -u3];



if check == 1
    Phi = Points(5).rP - Points(6).rP;


elseif check == 2
    rhsV = [0
            0];

elseif check == 3
    dcut1d = Points(5).rP_d;
    dcut2d = Points(5).rP_d –
                        Points(4).rP_d;
    rhsA = -s_rot(dcut1d*theta_d(1) +
                        dcut2d*theta_d(2));
end
end
```

Now wehave all the files to simulate the response of this system.

Next example shows how to incorporate a driver constraint in a model.

Example C.3 (closed-chain and a driver)

In this example we add a driver constraint to the preceding slider-crank mechanism. The driver represents a constant angular speed motor that acts about the pin joint *O*. The constructed M-files for this example can be found in `Models/SCd`. Most of the input M-files are the same as those in the `SC` model. In the following we show only the M-files that have been revised.

Since we have cut a revolute joint and a driver constraint, the model contains three constraints.

```
function inConst
    include_global

    nConst = 3;
```

The driver function for this example is $\theta_1 = \pi/3 + 2\pi t$. We add this constraint and its corresponding entities to the cut-joint constraints in the following function M-file.

The added entities for the driver constraints are:

(a) *Jacobian matrix* **C**

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

(b) *Coordinate constraints*

$$\theta_1 - \pi/3 - 2\pi t = 0$$

(c) *Right-hand side array of velocity constraints*

$$\dot{\theta}_1 = 2\pi$$

(d) *Right-hand side array of acceleration constraints*

$$-\dot{\mathbf{C}}\dot{\boldsymbol{\theta}} = 0$$

```
function CutJoint(check, theta, theta_d, t)

    ...
    C = [s_rot(dcut1)  s_rot(dcut2)   -u3
         1             0               0];

if check == 1
    ...
    Phi = [(Points(5).rP - Points(6).rP)
           (theta(1) - pi/3 -2*pi*t)];

elseif check == 2
    rhsV = [0
            0
            2*pi];

elseif check == 3
    ...
    rhsA = -[s_rot(dcut1d*theta_d(1) +
                          dcut2d*theta_d(2))
               0];
end
```

We now have all the files to simulate the response of this system.

C.4 POSTPROCESSING

The program `dap` simulates the dynamic response of a system by integrating the equations of motion from the initial time $t = 0$ to the final time. The program uses the MATLAB's integrator `ode45`, which reports the results at every Δt seconds (time period) as specified by the user[1]. At every reporting time step, the integrator saves the time in an array named `T`, and saves the joint coordinates and velocities for all the open-chain joints in an array named `uT`. None of the other results, such as joint accelerations and Lagrange multipliers associated with the cut joints or drivers, are saved. The joint coordinates and velocities are saved column-wise. For example, if we integrate the equations of motion of the sliding-pendulum system that contains two joints for 4.0 seconds, and ask for $\Delta t = 0.02$ reporting intervals, the arrays will be formed as listed in Table C.2.

The program `post` takes the computed values for the joint coordinates and velocities from `uT` at each reporting time step. It recovers the missing results by solving the equations of motion again at every time step, and saving all the results in several arrays as listed in Table C.3. The user should refer to these arrays for performing further analyses with the simulated results.

[1] The reporting time step is not the same as the integration time step. Integration time step used by `ode45` is variable and most likely much smaller than the reporting time step.

Table C.2 Arrays `T` and `uT` contain the output from the integrator `ode45`

t
0.00
0.02
...
3.98
4.00

θ_1	θ_2	$\dot{\theta}_1$	$\dot{\theta}_2$
1.00	0.52	0.00	0.00
0.99	0.52	-0.00	-0.13
	. . . .		
1.06	-0.01	-0.45	1.32
1.05	0.02	-0.48	1.37

Table C.3 List of arrays constructed by the program `post`

Array	**Size**	**Description**
`theta`	`nt × nJC`	Joint coordinates
`theta_d`	`nt × nJC`	Joint velocities
`theta_dd`	`nt × nJC`	Joint accelerations
`r`	`nt × nB × 2`	Translational body coordinates
`rd`	`nt × nB × 2`	Translational body velocities
`p`	`nt × nB`	Rotational body coordinates
`pd`	`nt × nB`	Rotational body velocities
`rP`	`nt × nP × 2`	Coordinates of points
`rPd`	`nt × nP × 2`	Velocity of points
`Jac`	`nt × nConst × nJC`	Jacobian matrix
`Lam`	`nt × nConst`	Lagrange multipliers

`nt`	Number of time steps (rows) including $t = 0$
`nB`	Number of bodies
`nP`	Number of points
`nConst`	Number of constraints
`nJC`	Number of joint coordinates

C.5 APPLICATION EXAMPLES

In addition to the three examples that we have already discussed for open and closed chain systems, several other examples are provided in this package. All of the models reside in the folder `Models`. Some of these additional models are discussed in Chapter 8 of the textbook and in the user manual for the program `DAP_BC`. In the following, we provide a very short comment for each of these models. It is recommended that the reader should simulate each model for a few seconds and then observe the animation of the simulated response. This should provide a simple visual description of each model. Detailed construction of each model is similar to the ones in Examples C.1, C.2, and C.3.

Double A-Arm Suspension (DAP_JC/Models/AA)

This model was discussed in detail in Section 8.1.1 of the textbook for modeling with `DAP_BC`. To model this system for `DAP_JC` simulation, the pin joint at *B* is cut resulting in two constraints in this model.

MacPherson Suspension (DAP_JC/Models/MP_A, MP_B)

These models were discussed in detail Section 8.1.2 of the textbook for modeling with DAP_BC.

MP_A: In this model that contains three moving bodies, the translational joint is cut resulting in two constraints.

MP_B: In this model that consists of two moving bodies, the revolute-translational joint is cut. This cut joint requires only one constraint equation.

Cart (DAP_JC/Models/Cart_C)

In Section 8.1.3 we discussed a simple example of a cart to demonstrate how to model a disc (wheel) rolling on the ground without slip. Four versions of the model were presented with different types of drivers. To demonstrate how to model a rolling disc without slip in DAP_JC program, we have selected one of the models, Cart_C.

The model contains three bodies, two revolute joints, and two wheels that roll on the flat ground without slip. Since the system is a closed chain, we cut the rolling disc on the front wheel; i.e., between body (*3*) and the ground, to form a cut open-chain system. Therefore we have one rolling disc in the forward kinematics of the open tree between body (*1*) and the ground, and one rolling disc that is cut and must be represented by constraints. A review of the M-files ChainOpen and CutJoint should clarify these concepts.

Rod Impacting the Ground (DAP_JC/Models/Rod)

A rod impacting the ground repeatedly was discussed for modeling with DAP_BC in Section 8.1.5. The same example is considered here for modeling with DAP_JC. Since there are no kinematic joints in this system, the joint coordinate model is practically identical to the body coordinate model.

Two Connected Rods Impacting the Ground (DAP_JC/Models/Rod2)

This example represents two rods that are connected by a pin joint drop under the force of gravity and contact the ground. This model allows three points to be candidates for impacting the ground.

Triple Pendulums (DAP_JC/Models/TP)

The triple pendulum shown in Figure 5.7 of the textbook (with relative joint coordinates) is modeled in this example with DAP_JC. This is an open chain system that does not require cutting any joints.