Design, Simulation and Virtual Testing

madymo

Programmer's Manual | VERSION 7.7

www.tassinternational.com

MADYMO Manuals

An overview of the MADYMO solver related manuals is given below. From Acrobat Reader, these manuals can be accessed directly by clicking the manual in the table below. Manuals marked with a star (*) are also provided in hard-copy (major releases only).

Theory Manual	The theoretical concepts of the MADYMO solver.
Reference Manual*	Detailed information on how to use the MADYMO solver and how to specify the input.
Model Manual*	Dummy, Dummy Subsystem and Barrier Models with simple examples.
Human Model Manual	Human Models and applications that make use of Human Models.
Tyre Model Manual	Documentation about Tyre Models.
Utilities Manual	User's guide for MADYMO/Optimiser, MADYMO/Scaler, MADYMO/Dummy Generator, MADYMO/Tank Test Analysis
Folder Manual	Describes the use of MADYMO/Folder.
Programmer's Manual	Information about user-defined routines.
Release Notes	Describes the new features, modifications and bug fixes with respect to the previous release.
Installation Instructions	Description for the system administrator to install MADYMO.
Coupling Manual	Description of coupling with ABAQUS, LS-DYNA, PAM CRASH/SAFE and Radioss and the TCP/IP coupling with MATLAB/Simulink.

TASS International provides extensive and high quality support for its products to help you in utilizing the software most efficiently. TASS International offers extensive hotline support for our software products, MADYMO, PreScan and Delft-Tyre. Our hotline support can be reached over phone as well as via email and will assist you with your questions regarding our different software products. Your requests will be dealt with in a fast and effective manner to support you in the continuation of your work in progress. On the website you will find your local representative with the accompanying support contact details.

Contents

1 Program set-up

This manual is intended for the experienced MADYMO user who wants to develop his own program modules. Some background information about the MADYMO program package is given for those users who intend to incorporate their own force interaction models and/or special input/output routines.

The MADYMO package is mainly written in FORTRAN. It consists of a small main program that performs calls to subroutines followed by all these subroutines. The program needs an input data file and produces several output files during the simulation. These output files can be used by post-processing programs to produce graphs and tables of the model results, or to perform additional calculations.

The user is allowed to design and add his own routines to the MADYMO package. For reasons of maintenance, entries are made for these user-defined routines which can be divided into:

MADYMO 3D	USINTFTNO	control routine
	USRJ13TNO	kinematic joint routines
	USRJ23TNO	
	USRJ33TNO	
	USRJ43TNO	
	USRJ53TNO	
	USRRD3TNO	routine for defining a road profile
	USRSY3TNO	unit numbers offset, control activation and initialisation of kinematic joint routines

The calls to these routines are not built in the MADYMO/Solver program. The MADYMO/Solver program checks at runtime if the user defined library (libuserdef.so) is present in the directory were the main xml file is run. User-defined input can be given in a separate file referred by the USER_FILE attribute of the MADYMO element.

In general user-defined modules are developed for special purposes. It is also possible to rewrite special user-defined routines to general purpose routines. TASS International BV can assist you to make them part of your MADYMO program. To keep the overview of your package create a sub-directory for every new user-defined application.

The total program set-up is briefly summarized in Fig. 1.1.

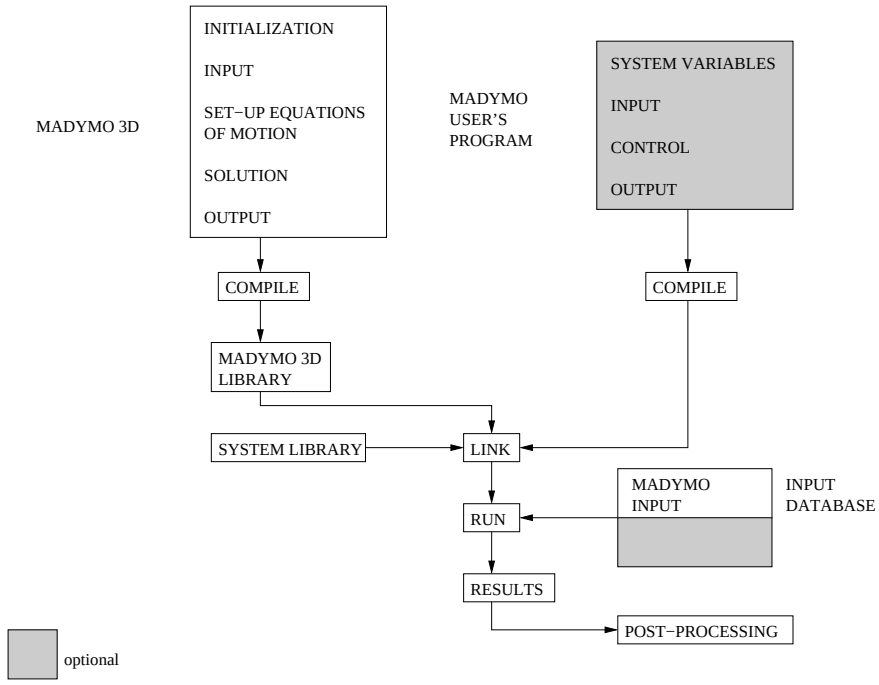

Figure 1.1: MADYMO Program set-up. Starting with R7.7, the MADYMO/Solver program checks the linking stage at runtime.

2 General

The user is allowed to design and add his own routines to the MADYMO package. This is, however, not supported on Windows system. In the .xml input file, user-defined input can be referred by the USER_FILE attribute of the MADYMO element. USER_FILE refers to a file in which the user-defined input is placed.

MADYMO uses a file table to manage the usage of logical unit numbers. These unit numbers are taken from the reserved range LUNOFF+1 to LUNOFF+TBLESZ (see subroutine USRSY3TNO in $MADAPP/userdef/usrsy3.f). The user may shift the range of unit numbers used by MADYMO to prevent conflicts with unit numbers used exclusively in the user routines.

2.1 Creating the user defined library libuserdef.so

To incorporate user subroutines in the MADYMO solver, fortran source code must be compiled and linked with MADYMO supplied libraries to create a new user defined routines library. The steps required to create the new library are given below.

1. Place all user defined fortran source files in a temporary directory, and change to that directory. Make sure only source files exist in this directory, and that the fortran compiler will recognise the files as source files (i.e. *.f). Example source files are located in <madymo_dir>/share/appl/userdef. Make sure that usrsy3.f is always included in the source files.
2. Create a shell with the MADYMO environment by executing the command:

   madymo77 -sh
3. Execute the command

   <madymo_dir>/share/run/makeexec.sh *.f

   This will attempt to compile the source files, and link them into the user-defined library libuserdef.so.
4. Check that a dynamic library libuserdef.so was successfully created.
5. Copy the library libuserdef.so into the directory where you want to run the simulation.
6. Exit from the MADYMO environment with the command

   exit

2.2 Running with user defined routines

Start the solver job with the command

madymo77 <input file>

With this method, the MADYMO/Solver locates the dynamic library in the current directory, and uses it as the user defined library for the job.

3 MADYMO library

In this chapter some general subroutines are described in detail. When the user develops his own joint or force model routines or special output routine(s), using these general subroutines is recommended. In the description of the modules (I) means integer, (D) double precision and (C) means character.

3.1 FILFG3TNO

Purpose:	Subroutine called to apply a force on a body		
Usage:	CALL FILFG3TNO (MODEL, NO, BOD, 0, XP, YP, ZP, FX, FY, FZ)		
Parameters:	None		
Input:	MODEL	(I)	Code for model
	NO	(I)	Code for identification of interaction
	BOD	(I)	Body number i.e. internal body number
	XP, YP, ZP	(D)	Point of application of the force, expressed in the local coordinate system of body BOD
	FX, FY, FZ	(D)	Components of the force, expressed in the local coordinate system of body BOD
output	none		

NOTES:

1. See Table A.10 of the Reference Manual for a more detailed description of table `FF`.
2. The variables `MODEL` and `NO` are only used for identification purposes in the file `DEBUG`. For user-defined force models `MODEL` 23 is recommended.
3. When action and reaction forces are calculated in a user-defined force model, subroutine `FILFG3TNO` must be called twice.
4. If necessary, subroutine `TRANSGTNO` can be used to calculate (`XP`, `YP`, `ZP`) and (`FX`, `FY`, `FZ`) in the local coordinate system of body `BOD`.

3.2 FILMG3TNO

Purpose:	Subroutine called to apply a torque on a body		
Usage:	CALL FILMG3TNO (MODEL, NO, BOD, 0, MX, MY, MZ)		
Parameters:	None		
Input:	MODEL	(I)	Code for model
	NO	(I)	Code for identification of torque
	BOD	(I)	Body number i.e. internal body number
	MX, MY, MZ	(D)	Components of the moment vector, expressed in the local coordinate system of body BOD
output	none		

NOTES:

1. See Table A.11 of the Reference Manual for a more detailed description of table `MM`.
2. The variables `MODEL` and `NO` are only used for identification purposes in the file `DEBUG`. For user-defined joint models `MODEL` 23 is recommended.
3. When action and reaction torques are calculated in a user-defined joint model, subroutine `FILMG3TNO` must be called twice.
4. If necessary, subroutine `TRANSGTNO` can be used to calculate (`MX`, `MY`, `MZ`) in the local coordinate system of body `BOD`.

3.3 TRANSGTNO

Purpose:	Transformation of the coordinates of a given local point P connected to body BOD1 (if BOD1 = 0 a point of the reference space) to • the local coordinate system of body BOD2 or • a coordinate system with the orientation of the local coordinate system of body BOD2 (if BOD2 = 0 the reference coordinate system), but with the same origin as the coordinate system of body BOD1.		
Usage:	CALL TRANSGTNO(IPAR, BOD1, X1, BOD2, X2)		
Parameters:	None		
Input:	IPAR	(I)	Selection parameter
	BOD1	(I)	Body number or reference space if BOD1 = 0
	X1	(D)	Array containing in X1(1), X1(2) and X1(3) the X, Y and Z coordinates of point P in the coordinate system of body BOD1
	BOD2	(I)	Body number or reference space if BOD2 = 0
output	X2	(D)	Array containing in X2(1), X2(2) and X2(3) the calculated X, Y and Z coordinates of point P expressed in: • the local coordinate system of body BOD2 (IPAR = 1) • a coordinate system with the orientation of the local coordinate system of body BOD2, but with the same origin as the local coordinate system of body BOD1 (IPAR = 2)

NOTES:

1. Figure 3.1 illustrates the function of subroutine `TRANSGTNO`. Point P is expressed in the coordinate system of body I. The coordinates of point P in the coordinate system of body J are found by:

   ```
   CALL TRANSGTNO(1, I, XI, J, XJ)
   ```

   The coordinates of point P expressed in the coordinate system (x_i', y_i', z_i') (orientation the same as orientation local coordinate system body J) are found by:

   ```
   CALL TRANSGTNO(2, I, XI, J, XI1)
   ```

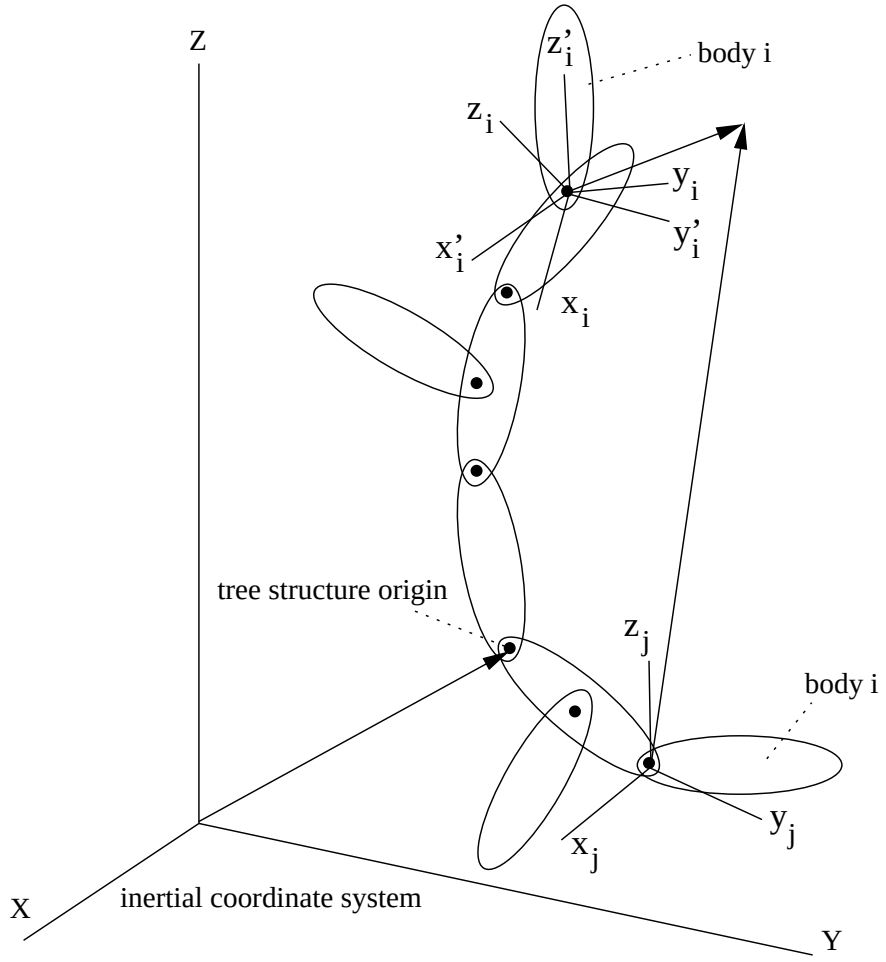

Figure 3.1: Illustration of the function of subroutine TRANSGTNO.

3.4 VELACGTNO

Purpose:	Calculation of the linear velocity or linear acceleration of a given local point P, expressed in the inertial coordinate system.		
Usage:	CALL VELACGTNO(IPAR, BOD, X, V)		
Parameters:	None		
Input:	IPAR	(I)	Selection parameter for linear velocity (IPAR = 1) or linear acceleration (IPAR = 2)
	BOD	(I)	Body number or reference space if BOD = 0
	X	(D)	Array containing in X(1), X(2) and X(3) the X, Y and Z coordinates of point P in the coordinate system of body BOD (in the reference coordinate system if BOD = 0)
output	V	(D)	Array containing in V(1), V(2) and V(3) the components VX, VY and VZ of the velocity or acceleration of point P, expressed in the inertial coordinate system

NOTES:

1. `BOD` = 0 : velocity and acceleration are zero
2. If the components VX, VY, VZ of the calculated velocity (acceleration) must be expressed in the local coordinate system of body I, subroutine `TRANSGTNO` can be used

```
CALL TRANSGTNO(2, 0, V, I, VI)
```

3.5 GETIDTNO

Purpose:	Get identifier number from body or joint name		
Usage:	CALL GETIDTNO(TYPE, NAME, IDNR, CLEN, CDATA)		
Parameters:	None		
Input:	TYPE	(I)	Type of object (either 'BODY' or 'JOINT')
	NAME	(C)	Original full reference of the body or joint
output	IDNR	(I)	inernal numerical identifier of the body or joint
	CDATA	(C)	internal name identifier of the body or joint
	CLEN	(I)	length of the CDATA character string

3.6 GETPCBTNO

Purpose:	Get joint information: internal body and node numbers of both parent and child		
Usage:	CALL GETPCBTNO(JNTNR, BODY1, NODE1, BODY2, NODE2)		
Parameters:	None		
Input:	JNTNR	(I)	Internal joint number
output	BODY1	(I)	Parent body number of joint
	NODE1	(I)	Parent node number of joint
	BODY2	(I)	Child body number of joint
	NODE2	(I)	Child node number of joint

3.7 GETPTRTNO

Purpose:	Get position of mermoy location of specific record specified (SYMBOL)		
Usage:	NUMBER = GETPTRTNO (SYMBOL)		
Parameters:	None		
Input:	SYMBOL	(C)	The name of the symbol
output	NUMBER	(I)	The memory location in the specific array

NOTES:

1. For a list of valid SYMBOL names and for wich arrays they are valid, see subsection 4.3.4 and 4.6.2

3.8 WRREPFTNO

Purpose:	Write a message to the MADYMO REPRINT file		
Usage:	CALL WRREPFTNO(MESSAGE)		
Parameters:	None		
Input:	MESSAGE	(C)	Message string that is printed in the REPRINT file
output	None		

4 User-defined modules

4.1 Introduction

The user-defined modules are supplied as example routines, which are located in directory: `<madymo_dir>/share/appl/userdef`. The user is allowed to program and link his own routine(s) to the MADYMO package. The user-defined subroutines are:

- USINTFTNO
- USREL2TNO
- USREP3TNO
- USREP4TNO
- USREV8TNO
- USRJ13TNO
- USRJ23TNO
- USRJ33TNO
- USRJ43TNO
- USRJ53TNO
- USRRD3TNO
- USRMM3TNO
- USRMM4TNO
- USRMS4TNO
- USRMV1TNO
- USRMV4TNO
- USRMV8TNO
- USRSY3TNO

When writing your own subroutines or functions, do not define names for new subroutines, functions or common blocks ending with TNO, unless you provide them yourself.

4.2 Initialising user-defined routines

This routine must always be present in the user defined library. It handles:

- The lower bound of the logical file unit number range
- The activation of the user-defined control interface
- The number of degrees of freedom for the JOINT.USER elements

MADYMO uses logical unit numbers within the range `LUNOFF+1` and higher. The user may increase the range of logical unit numbers used by MADYMO to prevent conflicts with unit numbers used exclusively by the user-defined subroutines. By default the value of `LUNOFF` is 200. The user-defined control interface is activated when the integer `USRACT` is not equal to zero. The joint degrees of freedom of user-defined kinematic joints must be initialised here via the integer `JNTDOF` array.

4.3 User-defined control routines

The user-defined interface routine controls the initialisation, input, output, derivatives and termination of MADYMO. The initialisation and termination are called at the start en the end of the simulation, respectively, where the input, output and derivatives are called at each multibody time step. Activation is done in subroutine `USRSY3TNO`.

The interaction between MADYMO and the interface routine is defined using the `FLAG` argument. The simulation flow is broken down in the following steps:

- simulation initialisation
- `FLAG` = 0; user-defined initialisation
- simulation loop

  (while time <= time_end) {
  - `FLAG` = 1; user-defined output signals
  - `FLAG` = 2; user-defined derivates
  - `FLAG` = 3; user-defined input signals
  - integration of derivatives
  - time history and animation output
  - time = time + step }
- `FLAG` = 9; user-defined termination
- simulation termination

See the subroutine for further detail about the available inputs and outputs for each phase. The general usage of the various phases is as follows:

FLAG = 0 Initialisation at the start of the simulation. All user-defined input must be placed in a separate input file which filename is referred by the attribute USER_FILE in the MADYMO element. The required input can be given free format. Note that character strings must be placed between apostrophes according to the standard Fortran specifications.

FLAG = 1 Output on each time step. This phase allows the user to transfer output signal values from the SIGNAL.EXTERNAL_OUTPUT elements to the user-defined routine or to a file. It is recommended to limit the amount of data written to a file, for this reason the variable `LTH` is introduced. `LTH` equals `.TRUE.` when time history data is written.

FLAG = 2 Derivatives on each time step. User-defined derivatives, joint and force models can be placed here. The calculated forces (together with their point of application) and torques must be expressed in the body local coordinate system. For each calculated force

(subroutine FILFG3TNO) and torque (subroutine FILMG3TNO) must be called to place the calculated forces (torques) on the related body. If action and reaction forces (torques) are calculated, subroutines FILFG3TNO and FILMG3TNO must be called twice.

The standard MADYMO hysteresis options are not described here. However, the user can define his own hysteresis algorithm. MADYMO library routines can be used, e.g. for coordinate transformation calculations.

FLAG = 3 Input on each time step. This phase allows the user to transfer input signal values to SIGNAL.EXTERNAL_INPUT elements from the user-defined routine or from a file. It is recommended to handle control or external data export in this routine. One can request for termination when FLAG $\neq$ 9 by setting the argument RETVAL equal to -1.

FLAG = 9 Termination at the end of the simulation. This phase is called during the termination of MADYMO, allowing the user to write additional data and close all open files.

4.3.1 User-defined control interface example

The interface is explained by controlling an inverted pendulum. The special user defined library is build using the command:

```
makeexec.sh usintf.f usrsy3.f.
```

For this task, the user-defined files usintf.f and usrsy3.f are needed for handling the user-defined control interface functionality and the activation of the control interface, respectively.

The content of the file usintf.f and usrsy3.f as well as the routines being called are given below. Note that the source files and the model related input files are not part of the distribution. The usage of the subroutines is explained only.

4.3.2 Body - Joint configuration table

The relation between bodies and joint is defined via the configuration in the input model. The subroutine GETPCBTNO allows the user to request attachment point information (body and node) of both parent and child via the argument JNTNR.

4.3.3 Body and Joint identifier and name resolving

Both id and name of bodies and joints are resolved by using the subroutine GETIDTNO. On input, the argument NAME contains the original full reference of the body or joint, argument TYPE equals BODY or JOINT, respectively. The integer argument IDNR, represents the internal identifier where the string argument CDATA(1:CLEN) represents the name.

4.3.4 Memory storage

The record JNRDAT contains joint data. The positions of the specific memory locations in this record is defined by symbols. The symbol values can be retrieved with the function GETPTRTNO.

Table 4.1 list the symbol names that can be retrieved with this function.

Table 4.1: Joint specific data, with respect to the body local coordinate system and origin.

Symbol	Array length	Data type	Description
JNRN13	3	D	Position of origin of parent body
JNRN14	3	D	Position of origin of child body
JNRN15	9	D	Orientation of parent body coordinate system
JNRN16	9	D	Orientation of child body coordinate system
JNRN17	7	D	Position degrees of freedom
JNRN18	6	D	Velocity degrees of freedom
JNRN19	6	D	Acceleration degrees of freedom
JNRN20	3	D	Relative position of child body w.r.t. parent body
JNRN25	6	D	Reaction forces and torques on parent body
JNRN26	6	D	Reaction forces and torques on child body

The record BDRDAT contains body data. The positions of the specific memory locations in this record is defined by symbols. The symbol values can be retrieved with the function GETPTRTNO.

Table 4.2 list the symbol names that can be retrieved with this function.

Table 4.2: Body specific data, with respect to the global coordinate system and origin.

Symbol	Array length	Data type	Description
BDRN01	3	D	Mass
BDRN02	9	D	inertia matrix w.r.t. the local coordinate system
BDRN03	3	D	position of centre of mass
BDRN04	3	D	position of local origin
BDRN05	3	D	linear velocity of local origin
BDRN07	9	D	rotation matrix of local coordinate system
BDRN08	8	D	angular velocity of local origin

4.4 User-defined joints

A user-defined joint can be used to create a kinematic joint type which is not available in MADYMO. It requires specification of the motion, and its time derivatives, of a joint coordinate

system on the child body j relative to a joint coordinate system on the parent body i in terms of joint coordinates. The process of creating a user defined joint will be described with the help of the kinematic joint shown in Fig. 4.1 in which two points on body j are constrained to translate relative to body i along straight perpendicular guides.

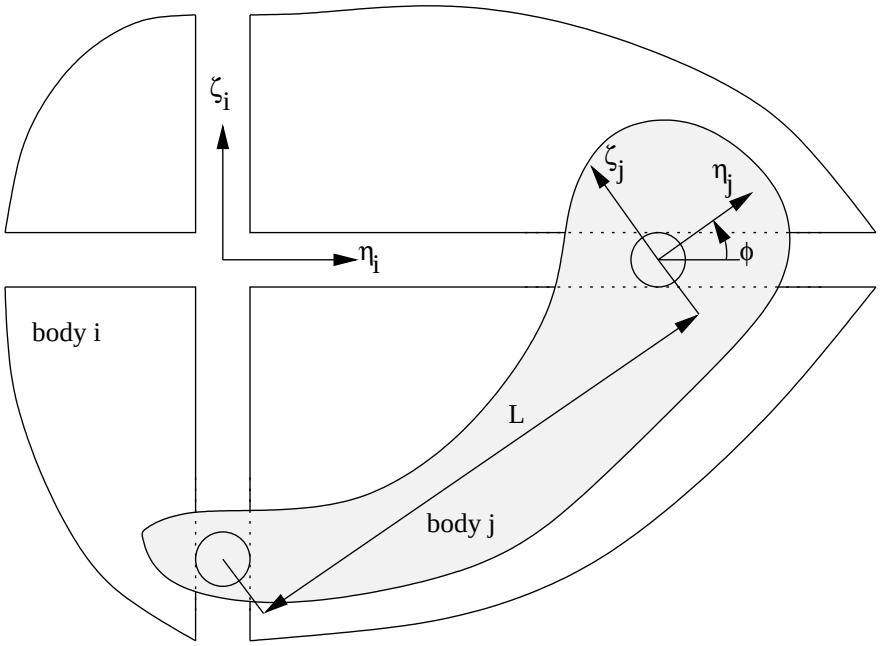

Figure 4.1: Example of a user-defined joint.

Joint coordinate systems may be chosen at will. It is recommended to choose them such that the expressions for the relative motion of the joint coordinate systems are simple in order to simplify the required preparations and to reduce the computation time. The origin of the joint coordinate system on body i is chosen coincident with the intersection of the two guides. The η_i- and ζ_i-axis are chosen parallel to the guides; the ξ_i-axis is perpendicular to the $\eta_i\zeta_i$-plane. The origin of the joint coordinate system on body j is chosen coincident with one of the two points that are constrained to translate along the guides. The η_j-axis is chosen parallel to the line that connects the two constrained points. The ζ_j-axis is parallel to the plane in which body j can move relative to body i. The ξ_j-axis is perpendicular to this plane.

This completes the definition of the joint coordinate systems. In the input, the orientation and the location of the origin of the joint coordinate systems have to be specified, in accordance with the above choice, in the same way as has to be done for standard joints.

The number of joint coordinates must equal the number of joint degrees of freedom. They should be chosen such that the expressions for the relative motion of the joint coordinate systems are simple. For this example, the angle ϕ between the η-axes of the joint coordinate

systems leads to simple expressions. The joint coordinates are assembled in a column matrix $\boldsymbol{q}$. For the current example this matrix becomes

$$\boldsymbol{q} = [\phi] \tag{4.1}$$

Next the motion of the joint coordinate system fixed to body j relative to the joint coordinate system fixed to body i must be specified. This must be in terms of the relative position vector of the origins of the joint coordinate systems and the relative rotation matrix.

The components of the relative position vector must be given with respect to the $\xi_i\eta_i\zeta_i$ system. This leads for the current example to

$$\boldsymbol{d}_{\mathrm{ij}} = \begin{bmatrix} 0 \\ L\cos\phi \\ 0 \end{bmatrix} \tag{4.2}$$

The relative velocity of the origins equals

$$\dot{\boldsymbol{d}}_{\mathrm{ij}} = \begin{bmatrix} 0 \\ -L\sin\phi \\ 0 \end{bmatrix} \dot{\boldsymbol{q}} = \boldsymbol{W}_T\dot{\boldsymbol{q}} \tag{4.3}$$

where the coefficient matrix $\boldsymbol{W}_T$ defines the axis of translation. The relative linear acceleration equals

$$\ddot{\boldsymbol{d}}_{\mathrm{ij}} = \begin{bmatrix} 0 \\ -L\sin\phi \\ 0 \end{bmatrix} \ddot{\boldsymbol{q}} + \begin{bmatrix} 0 \\ -L\dot{\phi}^2\cos\phi \\ 0 \end{bmatrix} = \boldsymbol{W}_T\ddot{\boldsymbol{q}} + \dot{\phi}^2\frac{\partial \boldsymbol{W}_T}{\partial \boldsymbol{q}} \tag{4.4}$$

The relative rotation matrix equals

$$\boldsymbol{D}_{\mathrm{ij}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{4.5}$$

Please note that the columns of the rotation matrix equal the components of unit vectors along, respectively, the ξ_j-, η_j- and ζ_j-axis with respect to the $\xi_j\eta_j\zeta_j$ coordinate system.

The partial derivative of the coefficient matrix $\boldsymbol{W}_T$ with respect to the joint coordinates equals

$$\frac{\partial \boldsymbol{W}_T}{\partial \boldsymbol{q}} = \begin{bmatrix} 0 \\ -L\cos\phi \\ 0 \end{bmatrix} \tag{4.6}$$

By definition, the relative angular velocity vector ω_{ij} is the axial vector of the skew-symmetric matrix $\dot{\boldsymbol{D}}_{\mathrm{ij}}\boldsymbol{D}_{\mathrm{ij}}^T$, i.e.,

$$\dot{\boldsymbol{D}}_{\mathrm{ij}}\boldsymbol{D}_{\mathrm{ij}}^T = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{4.7}$$

Substitution of (4.5) into (4.7) yields

$$\omega_{ij} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \dot{\boldsymbol{q}} \tag{4.8}$$

where the coefficient matrix $\boldsymbol{W}_R$ defines the axis of rotation,

$$\frac{\partial \boldsymbol{W}_R}{\partial \boldsymbol{q}} = 0. \tag{4.9}$$

Differentiation of (4.8) with respect to time yields the angular acceleration

$$\dot{\omega}_{ij} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \ddot{\boldsymbol{q}} \tag{4.10}$$

This completes the definitions and derivations which are necessary for a user defined joint.

Next, the kinematical equations must be introduced. The user can program up to 5 different subroutines, each with their own kinematic equations. These subroutines are named USRJ13TNO through USRJ53TNO. The first number (1 to 5) corresponds to the EXTERNAL_REF value of the JOINT.USER element. E.g. when it is equal to 2, the user has to program the subroutine USRJ23TNO. The arguments of these subroutine are given in Tab. 4.3.

On entry, all matrices that have to be filled, have already been initialized to zero except the diagonal terms of CD which have been initialized to unity.

An example of a user defined joint routine (USRJ13TNO) can be found in <madymo_dir>/share/appl/userdef.

Note that the variable SL, i.e. the distance L shown in Fig. 4.1, is assigned the value 0.15 in subroutine USRJ13TNO. It is also possible to read this value from the JOINT.USER element. This prevents changing the subroutine USRJ13TNO for another value of L.

In addition to creating this subroutine USRJ13TNO, the degrees of freedom of the new joint have to be specified in subroutine USRSY3TNO, the number of joint degrees of freedom has to be assigned to JNTDOF(1,1) and JNTDOF(2,1).

4.5 User-defined roads

The user defined subroutine USRRD3TNO can be used to create a road surface which cannot be specified by one of the standard road types. This subroutine is called if the ROAD.USER element is specified. The four geometric quantities R1, R2, R3 and R4, which can be specified by the ROAD_PAR attribute, can be used in subroutine USRRD3TNO as parameters in the definition of the road surface.

Table 4.3: Arguments of the user defined joint subroutine USRJ[1-5]3TNO.

Argument	Description
Q	This array contains the actual values of the joint coordinates $\boldsymbol{q}$
QT	This array contains the actual values of the first time derivative of the joint coordinates $\dot{\boldsymbol{q}}$
SD	This array must be filled with the components of the relative position vector $\boldsymbol{d}_{ij}$
SDT	This array must be filled with the components of the relative velocity vector $\dot{\boldsymbol{d}}_{ij}$
SDTT	This array must be filled with the part of the relative acceleration vector $\ddot{\boldsymbol{d}}_{ij}$ which does not depend on the second time derivative of the joint coordinates
SDQ	This array must be filled with the coefficient matrix of the second time derivative of the joint coordinates $\ddot{\boldsymbol{q}}$ in the expression for the relative acceleration
SDQP	This array must be filled with the partial derivative of the coefficient matrix $\boldsymbol{W}_T$ with respect to the joint coordinates. The last index in the three dimensional matrix represents the joint coordinate.
CD	This array must be filled with the relative rotation matrix $\boldsymbol{D}_{ij}$
CDT	This array must be filled with the components of the relative angular velocity vector ω_{ij} with respect to the $\xi_i\eta_i\zeta_i$ coordinate system
CDTT	This array must be filled with the part of the relative angular acceleration vector $\dot{\omega}$ which does not depend on the second time derivative of the joint coordinates
CDQ	Unused
CDQP	Unused

Subroutine USRRD3TNO requires the inertial z-coordinate of the road surface as a function of the inertial x and y coordinates, i.e. z(x,y), and the partial derivatives of this function with respect to x and y, i.e. $\partial z/\partial x$ and $\partial z/\partial y$. In calculating the point of contact between the tyre and the road, the road is approximated locally by a plane which is tangent to the road surface vertically below the wheel centre, i.e. in the direction of the negative z-axis of the inertial coordinate system. This is only a good approximation if the road is more or less parallel to the xy-plane of the inertial coordinate system and the radius of curvature of the road surface is large as compared to the radius of the tyre.

As an illustration of subroutine USRRD3TNO, consider the road profile defined by (see Fig. 4.2).

$$\begin{array}{lll} z = R_1 & \text{for} & x \le R_3 \\ z = R_1 - R_2\left(1 - \cos\left(2\pi\frac{x-R_3}{R_4}\right)\right) & \text{for} & R_3 < x < R_3 + R_4 \\ z = R_1 & \text{for} & x \ge R_3 + R_4 \end{array} \tag{4.11}$$

The corresponding partial derivatives of Z with respect to X and Y are given by

$$\begin{array}{lll} \frac{\partial z}{\partial x} = 0 & \text{for} & x \le R_3 \\ \frac{\partial z}{\partial x} = -2\pi\frac{R_2}{R_4}\sin\left(2\pi\left(\frac{x-R_3}{R_4}\right)\right) & \text{for} & R_3 < x < R_3 + R_4 \\ \frac{\partial z}{\partial x} = 0 & \text{for} & x \ge R_3 + R_4 \\ \frac{\partial z}{\partial y} = 0 & \text{for} & -\infty < x < \infty \end{array} \tag{4.12}$$

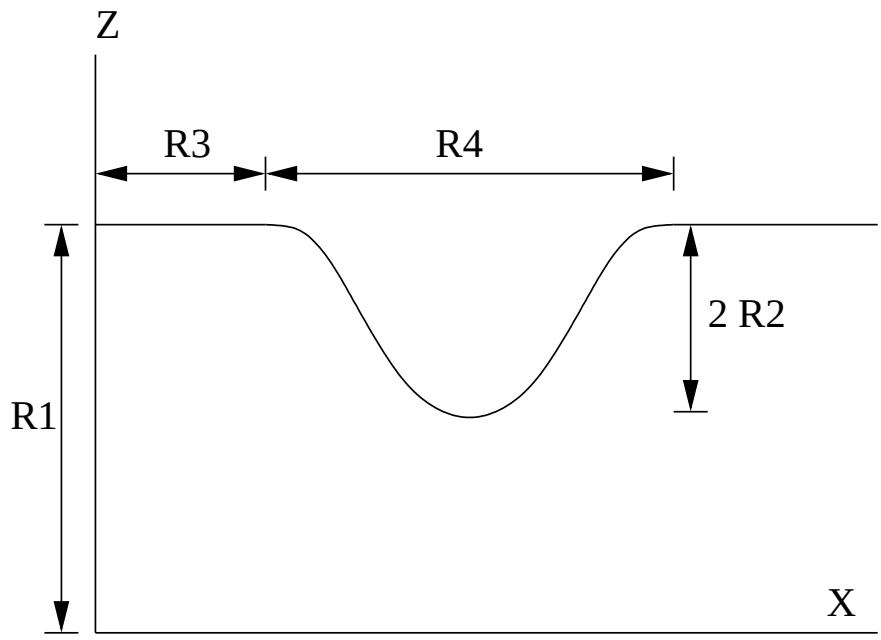

Figure 4.2: Illustration of a road with a cosine-shaped hole

4.6 User-defined FE-materials

Warning: The use of this option generally requires considerable expertise. The user is cautioned that the implementation of any realistic constitutive model requires extensive development and testing. Initial testing on a single element with prescribed traction loading is recommended.

User-defined materials for 3-node membrane, 4-node membrane, 4-node shell, 4-node tetrahedal and 8-node hexahedral can be implemented by creating subroutines for the material model in combination with MATERIAL.USER in the input. The options for specification of the material parameters in the MADYMO input deck are discussed in Sec. 4.6.1. Section 4.6.2 explains the use of the data records and pointers. The implementation of the material routines is discussed in Sec. 4.6.3.

4.6.1 Input

MADYMO allows the user to specify a material model called MATERIAL.USER for elements of type `TRIAD3`, `QUAD4`, `TETRA4` and `HEXA8`. When this material is selected, 30 material parameters may be specified by the user. The user can specify the different arguments by using the attribute MAT_PAR followed by the values of the parameters. An example of a user material

for an 8-node hexahedral element is given below.

4.6.2 Memory storage

The record PROPER contains material data. The positions of the specific memory locations in this record is defined by symbols. The symbol values can be retrieved with the function GETPTRTNO. Table 4.4 lists the symbol names that can be retrieved with this function.

Table 4.4: Non-element specific data

Symbol	Array length	Data type	Description
PRPSOS		D	speed of sound through the material
PRPUSR	30	D	user specified material parameters
PRPDEN		D	material density

4.6.3 User routines

Table 4.5 shows the user materials available in MADYMO. The files can be found in the directory <madymo_dir>/share/appl/userdef.

Table 4.5: User materials with subroutine names and element and property types.

Subroutine name	File name	Element type	Property type
USRMM3TNO	usrmm3.f	TRIAD3	MEM3/MEM3NL
USRMM4TNO	usrmm4.f	QUAD4	MEM4/MEM4NL
USRMS3TNO	usrms3.f	TRIAD3	SHELL3
USRMS4TNO	usrms4.f	QUAD4	SHELL4
USRMV1TNO	usrmv1.f	HEXA8	SOLID8 (FULL_INT="OFF")
USRMV4TNO	usrmv4.f	TETRA4	SOLID4
USRMV8TNO	usrmv8.f	HEXA8	SOLID8 (FULL_INT="ON")

These routines must be modified for the user-defined materials. The routines have an argument FLAG which indicates what the routine should do, i.e.

- **FLAG=1**

  The user subroutine must specify the lengths for the items that are stored in ELEDAT (element data array). Currently 9 items are available for the user and for every item the user should specify the number of positions in the array ELEDAT that is needed for that item

ELDUS1 pointer for user specified data (length = LUS1)
ELDUS2 pointer for user specified data (length = LUS2)
ELDUS3 pointer for user specified data (length = LUS3)
....
ELDUS9 pointer for user specified data (length = LUS9)

LUS_i are the number of positions in ELEDAT which are allocated for that pointer. These must be greater or equal than zero. Only if FLAG = 1 the lengths of the items can be changed.

Example:

```
C...    10 positions for ELDUS1 ->
C...    ELEDAT(ELEMNR,ELDUS1+J), J = 1, 10 can be used.
      LUS1 = 10
C...    5 positions for ELDUS2 ->
C...    ELEDAT(ELEMNR,ELDUS2+J), J = 1, 5 can be used.
      LUS2  = 5
C...    0 positions for ELDUS3-ELDUS9
      LUS3 = 0
      LUS4 = 0
      LUS5 = 0
      LUS6 = 0
      LUS7 = 0
      LUS8 = 0
      LUS9 = 0
```

- **FLAG=2**

  Initialisation of element data. FLAG=2 is called once per simulation at the start. The elements should be initialised during FLAG=2.

  For all user materials initialisation of PROPER(PRPSOS) is required. This is the speed of sound of the material and is used for calculation of the critical time step (Courant criterion) of the elements with the user materials.

  For some user materials, extra initialisation is required. For 8-node hexahedral elements with reduced integration (USRMV1TNO), 4-node membrane elements with reduced integration (USRMM4TNO) or 4-node shell elements (USRMS4TNO) it is required that the hourglass stiffness ELEDAT(ELEMNR,ELDHST) is specified. The hourglass stiffness is used for the calculation of the hourglass stabilisation forces.

  For a linear elastic isotropic material the modulus of elasticity can be used successfully. This is demonstrated in the example routine USRMV1TNO, USRMM4TNO and USRMS4TNO. However, when dealing with strong non-linear material behaviour, resulting in a large variation of the material stiffness, some experimentation with the optimum stiffness specified by the programmer may be necessary. The hourglass stiffness specified by the programmer may be updated every integration step.

- **FLAG=3**

  Execution phase. During FLAG=3 the stresses and energy of the materials must be calculated.

  The FE-integration cycle consists of four steps. In the first step the element strains (and incremental strains) are computed from the nodal displacements. The second step deals with the computation of the element stresses using the elements strains, or strain increments. With these stresses the nodal forces and moments are obtained during the third step. Finally the fourth step is the time integration. The nodal accelerations are calculated from the current nodal forces and nodal masses. The nodal velocities and nodal displacements are updated using these accelerations.

  When implementing a new material the actual material model is realised in the calculation of the stresses due to the strains. Both strains and incremental strains are available. The material model should result in stresses which are dual with the strains mentioned. For HEXA8 and TETRA4 elements the strains and stresses are expressed in the reference space coordinate system. For all other elements the strains and stresses are expressed in the local coordinate systems of the elements.

  The total internal energy and the dissipated energy due to the user materials should also be calculated in the user routines. Note that the elastic energy is the total internal energy minus the dissipated energy. If this is not done in a correct way, the energy balance in the energy output will be incorrect.

  In the supplied user routines with the MADYMO distribution, linear isotropic material behaviour is implemented as an example.

4.7 User-defined FE-elements

Warning: The use of this option generally requires considerable expertise. The user is cautioned that the implementation of any finite element requires extensive development and testing. Initial testing on a single element with prescribed traction loading is recommended.

Table 4.6 shows the user elements available in MADYMO. The user files mentioned there (`usrel2.f`, `usrep3.f`, `usrep4.f` and `usrev8.f`) can be found in the directory: <madymo_dir>`/share/appl/userdef`. Under the PART XML element, the element type is linked using material reference and property reference as specified in Tab. 4.6 for user-defined elements. An example of an eight-node hexahedral element is given below:

The property definition must be specified under the corresponding PROPERTY element. Material data must be defined under the MATERIAL.USER element (see Sec. 4.6).

The subroutines USREL2TNO, USREP3TNO, USREP4TNO and USREV8TNO must be modified for the user-defined element. The routines have an argument FLAG which indicates what the routine should do, i.e.,

Table 4.6: User FE-elements with subroutine names and property name.

Element type	Property type	Subroutine name	File name
2-node line	USERL2	USREL2TNO	usrel2.f
3-node line	USERL3	USREL2TNO	usrel2.f
3-node plane	USERP3	USREP3TNO	usrep3.f
4-node plane	USERP4	USREP4TNO	usrep4.f
8-node volume	USERV8	USREV8TNO	usrev8.f

FLAG = 1 The user subroutine must specify the lengths for the items that are stored in ELEDAT (element data array). Currently 13 items are available, and for every item the user should specify the number of positions in the array ELEDAT that is needed for that item:

ELDTST pointer for time step (LTST)
ELDBLK pointer for bulk modulus of element (LBLK)
ELDTHK pointer for thickness of element (LTHK)
ELDMAS pointer for mass of element (LMAS)
ELDUS1 pointer for user specified data (LUS1)
ELDUS2 pointer for user specified data (LUS2)
....
ELDUS9 pointer for user specified data (LUS9)

LTST, LBLK, LTHK, LMAS, LUSi are the number of positions in ELEDAT which are allocated. These must be greater than or equal to zero.

LTST must always be set on 1.

LBLK and LTHK must be defined if contacts of type FE_FE or MB_FE are used. LTHK does not have to be defined for USERL2 and USERV8 elements. LMASS must be defined if acceleration loads are used. The lengths (LTST, LBLK, LTHK etc.) must be specified per material (not per element). The user routines are called per material (MATNR is the material number).

Only if FLAG = 1 the lengths of the items can be changed.

Example:

```
C...   1 position for ELDTST
       LTST = 1
C...   1 position for ELDBLK
       LBLK = 1
C...   1 position for ELDTHK
       LTHK = 1
C...   1 position for ELDMAS
       LMAS = 1
C...   10 positions for ELDUS1
C      ELEDAT (ELMNR, ELDUS1+J), J = 0, 9 can be used.
       LUS1 = 10
C...   0 positions for ELDUS2-ELDUS9
```

```
LUS2 = 0
LUS3 = 0
LUS4 = 0
LUS5 = 0
LUS6 = 0
LUS7 = 0
LUS8 = 0
LUS9 = 0
```

FLAG = 2 Initialization of element data.

FLAG = 2 is called once per material at the starting time of the simulation. The elements must be initialized during FLAG = 2. The time step must also be calculated (ELEDAT(ELEMNR, ELDTST)) and the array with nodal masses must be updated for the user elements (NMASS and NRMASS). In TOPOL the topology of the elements is stored.

If necessary, the arrays ELEDAT(ELEMNR,ELDBLK), ELEDAT(ELEMNR,ELDTHK) and / or ELEDAT(ELEM must also be calculated (see FLAG = 1). Changing these values during FLAG = 3 has no effect on the simulation.

Note that the order of the elements and nodes are different in the subroutine in comparison with the input.

FLAG = 3 Execution phase.

During FLAG = 3 the nodal force must be calculated. The element nodal forces should be added to FNODE and FRNODE. For variable time step, ELEDAT(ELEMNR,ELDTST) can also be recalculated.

5 Compiler Requirements

MADYMO provides the ability to compile and link user defined Fortran routines. The requirements for the Fortran compiler are the following:

Platform-ID	**Platform**	**Fortran compiler version** *command for version*
		ifort -V
linux26-x86_64	Linux x86_64	PGI compiler pgf90 6.1-6 *pgf90 -V*

On linux, the user defined routines have been tested to work with g77/gfortran versions 3.2.3, 3.3.5 and 4.2.1.