MELODY 2D: Multibody ELement-free Open code for DYnamic simulations in 2D

USER GUIDE

Guilhem Mollon

Université de Lyon, LaMCoS, INSA-Lyon, CNRS UMR5259, 69621 Lyon, France

Last update: 24th of March 2019; Relative to versions 3.11 (preprocessor) and 3.87 (main code)

MELODY is a code dedicated to the simulation of multibody systems. It is primarily targeting applications in granular science, tribology, geomechanics and geophysics, but can be used for more general purposes. It is capable of handling a large number of rigid and/or (very) deformable bodies, and to have them interact by contacts. Its preprocessor is written in matlab, its main code is implemented in C++ in an open-MP framework, and its graphic outputs can be open with Paraview. Compiled versions of the main code do already exist for Windows and for Linux-based clusters, and it can in principle be compiled on Mac as well. In what follows, we describe the syntax of the preprocessing matlab files and the available material and contact laws, and we then detail the content of the text files used to input these data in the main code. The technical and scientific content of the code can be found elsewhere (DOI: 10.1002/nme.5258 and more recently DOI: 10.1007/s40571-018-0187-6).

I. Preprocessing scripts

The preprocessor is a matlab code containing several .m files. To generate data files for the main program, one must execute the main script "MELODY.m". This script should not be modified, apart from the line 23 which calls a secondary script (often called "MELODYLoadData_*MySimulation*"). This secondary file is the one which contains all the description of the future simulation, and it thus the one the user should be creating and/or modifying to make his/her modelling choices. In the remainder of this section, we thus focus only on this file, and we use the illustrative case provided in MELODYLoadData_Example.m. A typical .m data script is organized in several sections as follows (although there is no obligation to keep exactly this structure):

-Simulation name
-Initialization
-Bodies
-Materials and contact laws
-General boundary conditions
-Text outputs
-Graphic Outputs
-Numerical parameters

-Flags.
In the end, the script must provide a certain number of matlab variables, which are necessary for the execution of the preprocessor:

-Activate_Plot	(*double 0 or 1)*
-Alid	*(N_b×3 cell)*
-Bodies_Materials	*(N_b×1 cell)*
-Contact_Laws	*(N_{cl}×3 cell)*
-Contact_Updating_Period	(*double)*
-Contours	*(N_b×1 cell)*
-Deactivations	*(N_d×3 cell)*
-Detections	*(N_b×2 double)*
-Distributions	*(N_b×3 cell)*
-Gravity	*(2×1 double)*
-Imposed_Pressures	*(N_b×1 cell)*
-Imposed_Velocities	*(N_b×1 cell)*
-Initial_Velocities	*(N_b×1 cell)*
-Initialize_CZM	(*double 0 or 1)*
-Interpolations	*(N_b×2 cell)*
-Integrations	*(N_b×2 cell)*
-Materials	*(N_m×3 cell)*
-Mesh_Ratios	*(N_b×2 double)*
-Monitorings	*(N_{mo}×1 cell)*
-Periodic_Boundaries	*(1×2 double)*
-Save_Periods	*(1×2 double)*
-Scheme	(*string)*
-Scheme_Parameters	*(N_{pa}×1 double)*
-Simulation_Name	(*string)*
-Spies	*(N_s×4 cell)*
-Status	*(N_b×1 cell)*
-Time_Stepping_Parameters	*(1×3 double)*
-To_Plot	*(27×1 double)*

In these variable formats, N_b is the number of bodies in the simulation, N_m is the number of materials, N_{cl} is the number of contact laws, N_{mo} is the number of monitorings, N_s is the number of spies, and N_{pa} is the number of parameters of the chosen numerical scheme. Of course, additional variables may be created if needed. More generally, any functionality of matlab can be used in the script if it is necessary for the creation of the simulation data. External scripts or data can be loaded as well.

The structure of the main program of MELODY is based on a "bodies" logic, and the preprocessor reflects this structure. Basically, the purpose of the preprocessing script is to create a certain number of individual bodies, to provide their geometries, their discretization, their types, their mechanical properties, their boundary conditions, and the way they interact by contacts. Some additional data (such as the way the problem should be solved, what outputs are needed, etc.) are defined as well. We now detail each typical section of a preprocessing script.

1. Simulation name

In this section we only define one variable, **Simulation_Name** (*string)*, which is the name of the simulation (it will also be the name of the folder where the data files will be created).

2. Initialization

In this section, we just create some empty variables that are to be used later in the script.

3. Bodies

A large number of variables must be created in this section:

-Contours *(N_b×1 cell).* Each element of this *cell* is another *cell* which contains all the information in relation with the contour of a given body. Several types of bodies can be generated, and their contours are represented by a certain number of borders: '***Closed***' bodies have only one border, which closes on itself (example: a disc), '***Simple***' bodies may have several borders which altogether form a closed contour (example: a square defined by four borders, one for each side), and '***Periodic***' bodies have two borders (one lower and one upper) which run from the left-hand limit of the periodic space of the simulation to the right-hand one. Hence, the *cell* contains one row for each border, and each row has three columns (or possibly five) with the following information:

-a *string* giving the type of border ('***Closed***','***Simple***', or '***Periodic***')

-an array *(N_p×2 double)* giving the x- and y-coordinates of the N_p points defining the geometry of this border. If grains with complex geometries are to be defined, they can be imported (with the appropriate format) from the output of a *Packing2D* generation. See the documentation of this specific package for more details.

-a *string* giving the kind of interpolation used for this boundary (only '***Linear***' is available)

-two scalar values, which should only be provided if the body is of the class ***'Simple'***: Those are the indices of the previous and next borders (this is useful for complicated topologies, for example in the case of a hole).

There are a few rules: for '***Closed***' and '***Simple***' bodies, the nodes (and the borders, in the ***'Simple'*** case) should be given in a counterclockwise order. For the '***Periodic***' bodies, the lower border should be given first (with border points provided from left to right), and the upper one should be given afterwards (with border points provided from right to left). If the border nodes are very far from each other, the preprocessor will add some other nodes in-between by linear interpolation.

-Distributions (*N_b×3 cell).* This *cell* has three columns, and each row corresponds to a given body.

-The first column is a *string* which defines the type of distribution of the nodes, and it also defines the type of body. It may be '***Structured***' (the body will be deformable, and the nodes will be on a regular grid), '***Unstructured***' (the nodes will be distributed using the Distmesh

meshing tool, see http://persson.berkeley.edu/distmesh/ for details. The body will also be deformable), or '***Rigid***' (Distmesh will be used again, but this time the body will be rigid).

-The second column is a typical distance between two neighboring nodes in the body

-The third column is used in the case of large deformable bodies, for load balancing between CPUs (if several of them are to be used): it is the number of regions into which this body should be divided in order for its simulation to be distributed on several CPU. Typical values of 20 may be used for very large bodies, 1 is good for small ones if there are a lot of them.

-**Interpolations** *(N_b×2 cell)*. Each row of this *cell* corresponds to a body. If the body is rigid, the row is ignored.

-The first element of a given row is a *string* defining the type of interpolation that will be used to interpolate the displacement field between the nodes. It can be '***MLS***' (in that case, Moving Least Square meshfree shape functions will be used) or '***T3***' (in that case, 3-nodes linear finite elements will be used). '***MLS***' shape functions should be used if large deformations of the body are expected, because they will provide a lot of robustness compared to a distorted mesh. If the body remains in small deformations (even if large displacements and rotations are expected), '***T3***' may be preferred because it is a bit faster.

-The second element of a given row is the desired number of nodes in the circular domains used for each node if MLS shape functions are used. For T3 bodies, it is just ignored.

-**Integrations** *(N_b×2 cell)*. Each row of this *cell* corresponds to a body. If the body is rigid, the row is ignored.

-The first element of a given row is a *string* defining the type of numerical integration used in the corresponding body (only '***Gauss***' is available at the moment).

-The second element is the number of Gauss points to be used in each triangle composing the domain of the body. It may be 1, 3, 4, 6, 7, 12, 13 or 16. Three integration points are usually enough.

-**Detections** *(N_b×2 double)*. Each row of this *cell* corresponds to a body.

-The first one is a distance used for broad proximity detection

-The second one is used for close contact detection distance

Both values are typically of the order of magnitude of the nodal distance of the body, but should be chosen adequately (as well as the contact updating period) in order not to miss any contact.

-**Bodies_Materials** *(N_b×1 cell)*. Each row of this *cell* corresponds to a body, and contains a *string* which is the name of the material assigned to this body. This *string*, in turn, should appear in the list of the materials defined for the simulation (see further).

-**Imposed_Pressures** *(N_b×1 cell)*. Each row of this *cell* corresponds to a body, and contains another *cell* with all the information related to the possibleNeumann (pressure) boundary

conditions applied on the borders of this body. This *cell* contains four columns and a number of lines equal to the number of borders of the current body. The first two columns correspond to the condition along x, and the last two ones to that along y. Each couple of columns contains the history of the loading and the type of condition ('***None***','***Oriented***',or '***Following***'). If the j^{th} border of the i^{th} body has no Neumann boundary condition along a given direction, then the corresponding columns should be filled with an empty *array* [] and '***None***'. Otherwise, the history of this boundary condition is input as a $(N_{times}\times 2$ *double)*, with the first column containing the N_{times} sorted instants and the second column contains the corresponding imposed pressures (the pressure will be linearly interpolated between the instants during the simulation). '***Oriented***' means that the applied force will keep its direction, while '***Following'*** means that its orientation will follow the possible rotation of the boundary

-**Imposed_Velocities** $(N_b\times 1$ *cell).* Each row of this *cell* corresponds to a body, and contains another *cell* with all the information related to the possible Dirichlet boundary conditions applied on the borders of this body. This *cell* has six columns and a number of lines equal to the number of borders of the current body. The first three columns correspond to the condition along x, and the last three to that along y. Each batch of three columns contains, in that order, the history of the loading, the type of condition, and the possible parameters of this condition. If the j^{th} border of the i^{th} body has no Dirichlet boundary condition along a given direction, then the corresponding three columns should be filled with an empty *array* [], '***None***', and another empty *array* []. Otherwise, the history of this boundary condition is input as a two columns *array*, with the first column containing sorted instants and the second column contains the corresponding imposed velocities (the velocity will be linearly interpolated between the instants during the simulation). The two possible types of conditions (apart from '***None***') are '***Driven***' and '***Soft***'. The first one corresponds to a true Dirichlet boundary condition and is only applicable to rigid solids, while the second one corresponds to a penalized Dirichlet boundary condition. In the latter case, two parameters (a stiffness and a damping) should be provided in the third column. The damping is generally equal to 0, but the stiffness should be chosen adequately: if it is too small, the condition will not be ensured with great accuracy, but if it is too large it will lead to extremely small time steps.

-**Initial_Velocities** $(N_b\times 1$ *cell)*: Initial velocities of each body. This *cell* has one column and a number of lines equal to the number of bodies present in the simulation. Each line contains a $(1\times 2$ *double),* containing the initial velocities of the body along x and y.

-**Mesh_Ratios** $(N_b\times 2$ *double)*: Possible gradient in the density of field nodes for each body (only works for '***Periodic***' and '***Closed***' bodies, at the moment). This array-variable has two columns and a number of lines equal to the number of bodies present in the simulation. If the body is '***Periodic***', the first column corresponds to a ratio applied to the nodal distance of the body at its lower boundary (small y), and the second column corresponds to the same for the upper boundary (large y). If the body is '***Closed***', the first column corresponds to a ratio applied to the nodal distance of the body at its outer boundary, and the second column corresponds to the ratio applied at the center. Hence, in the first case it is a vertical gradient and in the second case it is a radial gradient. There is a special case for the ***'Polygon'*** body type. In this case, only the first column is accounted for, and this number describes a spatial gradient of density, from the contour towards the inside. A good value should be about 0.2 to 0.5.

-**Status** *(N_b×1 cell)*: For each body, the corresponding row of this *cell* contains a *string* specifying if it is '***Active***' or '***Inactive***'. In both cases the body will be generated, but in the latter it will be completely ignored during simulation.

-**Alid** *(N_b×3 cell)*: Absorbing boundary conditions. They are useful for two purposes: provide an overdamping to the motion of a Neumann boundary in the case of a deformable body (by the means of the beta parameter of the Rayleigh damping), or avoid reflection of elastic waves at a Dirichlet or Neumann boundary (using both the alpha and the beta parameters). In order to apply absorbing boundary conditions, one should consider a "box", defined by four scalars *xmin, xmax, ymin* and *ymax*. The alpha and beta Rayleigh parameters are then increased from the inside towards the outside (i.e. towards the limits of the box), on a certain distance *d*. For example, concerning the upper boundary, the alpha parameter is increased from *alpha_inside* (at y=*ymax-d*) to *alpha_outside* (at y=*ymax*), and likewise for beta (and for the other boundaries). In this framework, y corresponds to the initial positions of the nodes. This damping comes in addition to that used naturally for the considered material, and we therefore usually have *alpha_inside*=0 and *beta_inside*=0. The increase follows a power law with a given exponent (3 is a good value). If some boundaries are to be ignored by this boundary condition, simply put the box limits far away from the solid. For a proper application of the beta part of the damping, the prescribed motion of the body should be defined (otherwise the Alid penalizes a prescribed displacement).

Each row of this *cell* corresponds to a given body, and the 3 columns contain the following data for this body:

-The first element is a *(1×12 double)*:

[N_{times_x}, N_{times_y}, xmin, xmax, ymin, ymax, d, alpha_inside,

alpha_outside, beta_inside, beta_outside, exponent]

-The second element is the (possible) history of prescribed motion along the x-direction, is a Dirichlet boundary condition is applied to this solid. It can also be a velocity which is not prescribed, but which is wanted for the body (deviation from this velocity is thus penalized by the beta-parameter). If no such motion exist, an empty *array []* is used (and N_{times_x} should be put to 0 in the previous array). Otherwise, it is a *(N_{times_x}×2 double)* which contains a velocity history (just like in the "imposed_velocities" section).

-The third element is analogous to the previous one, but for the y-direction.

-**Deactivations** *(N_d×3 cell)*: This variable is to be used if some bodies are to be deactivated during simulation. Only to be used in very specific contexts.

4. Materials and contact laws

In this section, we define the physics of the mechanical behavior of the bodies and of their interactions.

-**Materials** *(N_m×3 cell)*: List of the materials used in the simulation. This *cell* has three columns and a number of lines equal to the number of materials used in the simulation. The first column contains a *string* with the material name (this is the name called by the **Bodies_Materials** variable), the second column contains a *string* with the material type (with respect to the materials database which is described further in this document), and the third column contains a *(1×N_{param} double)*, corresponding to the parameters of the material (see also the materials database). It should be noted that, for every type of material, the first three parameters are respectively the density, the alpha Rayleigh damping, and the beta Rayleigh damping.

-**Contact_Laws** *(N_{cl}×3 cell)*: List of the contact laws used in the simulation. This *cell* has five columns and a number of lines equal to the number of contact laws used in the simulation. The first two columns contain two *strings* with the names of the materials in contact (these names are called by the **Bodies_Materials** variable), the third column contains a *string* with the type of contact law (with respect to the contact laws database which is described further in this document), the fourth column contains a keyword ('***Fixed***' or '***Evolutive***'), and the fifth column contains a *(1×N_{param} double)*, corresponding to the parameters of the contact law (see also the contact laws database). '***Fixed***' means that the contact law acts as if the length of the segment was unchanged, while '***Evolutive***' means that the evolution of the border length is accounted for. It should be noted that, for every type of contact law, the first two parameters are respectively the normal stiffness and the tangential stiffness (which generally have the same value). These values should be chosen adequately: if too small, the contact will not be ensured with great accuracy, but if it is too large they will lead to extremely small time steps.

5. General boundary conditions

-**Periodic_Boundaries** *(1×2 double)*: Left-hand and right-hand boundaries of the periodic space. If no periodicity is needed, then these values should be chosen far enough from the system in order for it not to be disturbed by those boundaries (they cannot be ignored by the code).

-**Gravity** *(2×1 double)*: Horizontal and vertical gravity accelerations.

6. Text outputs

-**Monitorings** *(N_{mo}×1 cell)*: Used only for debugging, should be ignored.

-**Spies** *(N_s×4 cell)*: This variable specifies the virtual sensors that are positioned on the system during its simulation in order to extract relevant data while avoiding a complete saving of the

current state for further postprocessing. Each row of this *cell* corresponds to a given spy, which will write some data in a certain text file at a certain frequency. Hence, each row contains four successive elements:

-A *string* with the name of the file into which the data of this spy will be written.
-The number N_d of different data collected by this spy (for example 2 if the spy takes the x- and y- coordinates of a certain node).
-The period of collection of these data (in time units)
-A *(N_d×1 cell)* describing exactly what are the N_d variables that are to be collected. In each element of this cell, a single *string* contains the type and location of requested data.

11 kinds of outputs can be extracted, with the following syntax:

'***Position [X, Y, R, Length, LengthInContact, Area, AllX, AllY] body [node, -1]***'
'***Displacement [X, Y, R, Mag] body [node, -1]***'
'***Velocity [X, Y, R, Mag] body [node, -1]***'
'***Acceleration [X, Y, R, Mag] body [node, -1]***'
'***Force [X, Y, R, Mag] [Total, Internal, Contact, Body, Dirichlet, Neumann, Damping] body [node, -1]***'
'***Jacobian body node***'
'***Stress [XX, YY, XY, ZZ, Tresca, VM, Major, Intermediate, Minor, Spherical] body [node, -1]***'
'***Contact [Gapn, Gapt, Xsi, Xnorm, Ynorm, Damage, Fx, Fy, Length, Master, Px, Py, Slave, Xslave, Yslave] body border bordernode***'
'***Damage body***'
'***Energy [Kinetic, Deformation] [body, -1]***'
'***Work [Internal, Contact, Body, Dirichlet, Neumann, Damping, Alid] [body, -1]***'

Values between [] are the possible options for a given argument (exactly one should be chosen). All arguments are needed. "body" is the number of the body from which we want to extract the data, and "node" is the number of the node. If the value -1 is entered (where possible), then the quantity is averaged over all the elements (all the nodes of the body for "node", all the bodies of the system for "body"). "Mag" means magnitude. Successive keywords should be separated by a simple space.

7. Graphic outputs

-**To_Plot** *(27×1 double)*: This array contains either 1 or 0 at each of its 27 positions, depending if the corresponding quantity is to be written in a graphic file. Since this operation is quite slow, it is recommended to avoid writing useless data on the disc. The list of the quantities is:

1. Body Index
2. Initial Position
3. Current Position
4. Displacement
5. Velocity
6. Acceleration
7. Force
8. Internal Force
9. Contact Force

10. Body Force
11. Dirichlet Force
12. Neumann Force
13. Damping Force
14. Alid Force
15. Jacobian
16. Cauchy XX Stress
17. Cauchy YY Stress
18. Cauchy XY Stress
19. Cauchy ZZ Stress
20. Tresca Stress
21. Von Mises Stress
22. Major Principal Stress
23. Intermediate Principal Stress
24. Minor Principal Stress
25. Spherical Stress
26. Green-Lagrange XX strain
27. Green-Lagrange YY strain
28. Green-Lagrange XY strain
29. Norm of the Green-Lagrange strain tensor
30. Body Damage
31. Normalized Displacement Error
32. Displacement Error

8. Numerical parameters

-**Scheme** (*string)*: Defines the type of time-stepping scheme to be used for solving. At the moment, only '***Adaptive_Euler***' is available.

-**Scheme_Parameters** *(N_{pa}×1 double)*: Numerical parameters of the chosen scheme. For the Adaptive Euler solver, three parameters are needed:

-Target maximum normalized displacement error

-Exponent of the time step correction heuristic

-Error threshold (when multiplied to the target time step) above which the time step is rejected

A satisfactory set of parameters is [1e-4, 0.1, 10]. It should be noted that the set [1e20, 0, 1e20] leads to a fixed time step, which could be interesting in certain applications.

-**Contact_Updating_Period** (*double)*: Period at which the proximities between the bodies are updated (in time unit).

-**Time_Stepping_Parameters** *(1×3 double)*: This array contains successively the initial time of the simulation, the time step at the beginning of the simulation (which will change afterwards if the solver is adaptive), and the final time of the simulation. All are given in time unit.

-**Save_Periods** *(1×2 double)*: Periods at which data are written on the disc. The first one is for the saving of the complete state of the system (which can later be used to relaunch a simulation, or to extract data for postprocessing), and the second one is related to the writing of a graphic file.

9. Flags

-**Activate_Plot** (*double 0 or 1)*: Should be 1 if the user wants to see in real time the generation of the sample in graphic display (it makes it a bit slower), 0 otherwise.

-**Initialize_CZM** (*double 0 or 1)*: This flag is only used if CZM contact laws with damage are used in the simulation (see contact laws description for more details). It should be 1 if these contacts are to be initialized (no damage for active contacts, complete damage for the others), and 0 if their status should be left unchanged.

II. Physical models

1. Constitutive models

Only two constitutive models are available at the moment in the code, and both correspond to elastic materials. In any case, some damping is introduced using the Rayleigh formalism, with two coefficients: Alpha is the contribution of the stiffness matrix (it somewhat represents the reluctance of the material to deform rapidly), and Beta is the contribution of the mass matrix (it somewhat represents the reluctance of the material to move rapidly with respect to the main frame of the simulation). The latter is not objective and may be ignored in the general case (using a small value such as 1e-20).

- Linear elastic material

This model is only to be employed if the considered body is expected to remain in the regime of small deformations and small rotations. Basically, it is based on the Hooke law which states linearity between the tensor of the linearized strains and the tensor of the Cauchy stresses.

Keyword: '***ElasticLinear***'

Parameters: Density
Alpha (Rayleigh Damping)
Beta (Rayleigh Damping)
Young Modulus
Poisson Coefficient

- Neo-hookean material

This model belongs to the hyperelastic class. It makes it possible to apply elasticity (i.e. reversibility) to bodies which will be submitted to large deformations and/or large rotations.

Keyword: '***NeoHookean***'

Parameters: Density
Alpha (Rayleigh Damping)
Beta (Rayleigh Damping)
Young Modulus
Poisson Coefficient

2. Contact models

Several contact models are implemented in the current version of the code. For all the models, the first two parameters correspond to a normal and a tangential stiffness. They are all written in a penalization framework.

- **Frictionless contact**

The simplest model, only avoiding large interpenetration between the considered bodies.

Keyword: '***Frictionless***'

Parameters: k_n (normal stiffness)

- **Cohesive contact**

A model with an attractive law which acts against separation (up to a certain tensile normal stress) and against slip (up to a certain tangential stress). This law is permanent in the sense that, if surfaces get separated but meet again later on, the contact model will be active again.

Keyword: '***Cohesion***'

Parameters:

k_n	(normal stiffness)
k_t	(tangential stiffness)
g_{tang}	(maximum tangential stress before separation)
g_{tens}	(maximum tensile stress before slip)

- **Classical Mohr-Coulomb contact**

This is the model that should be used in the case of two bodies in frictional contact, if at least one of them is deformable. It also includes a tangential "cohesion" (which is added to the frictional contribution) and a tensile cut-off.

Keyword: '***MohrCoulomb***'

Parameters:

k_n	(normal stiffness)
k_t	(tangential stiffness)
fric	(friction coefficient)
coh	(tangential cohesive stress)
tens	(tensile cut-off stress)

- **Damped Mohr-Coulomb contact**

Identical to the previous model, but to be used in the case of a contact between two rigid bodies. It hence includes a damping term.

Keyword: '***DampedMohrCoulomb***'

Parameters:

k_n	(normal stiffness)
k_t	(tangential stiffness)
fric	(friction coefficient)
coh	(tangential cohesive stress)
tens	(tensile cut-off stress)
damp	(normalized damping)

- **Impact Mohr-Coulomb contact**

Similar to the previous model, it should be used when two rigid bodies are interacting by impact. It dissipates a part of the impact energy by using a smaller stiffness during unloading. For more details, see DOI: 10.1016/j.enggeo.2012.07.021.

Keyword: '***TwoSlopesMohrCoulomb***'

Parameters:

k_n	(normal stiffness)
k_t	(tangential stiffness)
fric	(friction coefficient)
coh	(tangential cohesive stress)
tens	(tensile cut-off stress)
rati	(ratio between the loading and unloading stiffnesses)

- **Degradable Mohr-Coulomb contact**

This is the simplest model for bond breakage. It contains an internal variable representing the damage of the bond, which can be either 0 (intact bond) or 1 (broken bond). The bond has a certain tensile stiffness (which may be different from the one used for penalization), and will follow two different Mohr-Coulomb behaviors depending if it is intact or broken. The status will switch from intact to broken if the "intact" Mohr-Coulomb criterion is violated in any way. Bonds can be initialized (or reinitialized) to the "intact" status if the keyword "InitializeCZM" is written in the DYNAMIC file used as the starting point of a simulation. A given contact will be initialized as "intact" if the normal gap is lower that a certain value gap_{init}.

Keyword: '***BondedMohrCoulomb***'

Parameters:

k_n	(normal stiffness)
k_t	(tangential stiffness)
k_{bond}	(intact bond tensile stiffness)
$fric_{bond}$	(intact friction coefficient)
coh_{bond}	(intact tangential cohesive stress)
$tens_{bond}$	(intact tensile cut-off stress)
$fric_{free}$	(broken friction coefficient)
coh_{free}	(broken tangential cohesive stress)
$tens_{free}$	(broken tensile cut-off stress)
damp	(normalized damping)
gap_{init}	(initial detection distance to set a contact as "intact")

- **Cohesive zone model**

This model is a bit more complex that the previous one regarding the damage law, but simpler when the bond has broken. Its internal damage parameter can take any value between 0 and 1, and it is increased when the current degradation criterion (maximal tangential or tensile gap) is reached. This criterion evolves with the damage (maximum gaps increase), as well as the tensile and tangential stiffnesses (which both decrease when the damage increases). When the bond is broken, the contact switches to a model similar to the "Cohesion" one. Bonds can be initialized (or reinitialized) to the "intact" status if the

keyword "InitializeCZM" is written in the DYNAMIC file used as the starting point of a simulation. A given contact will be initialized as "intact" if the normal gap is lower that a certain value gap_{init}.

Keyword: '***CZMlinear***'

Parameters:

k_{ini}	(intact normal stiffness)
P_{nlim}	(tensile cut-off stress)
gap_{nlim}	(tensile cut-off gap)
p_{nres}	(residual cohesion when the bond is broken)
nt_{ratio}	(ratio of all the tangential parameters to the normal ones)
gap_{init}	(initial detection distance to set a contact as "intact")

- **Cohesive zone model with fatigue**

This model is the most sophisticated damage model currently implemented in the code. It has the same features than the previous ones, but with an additional ingredient related to fatigue. The internal damage parameter can increase either because the current threshold is reached (in a somewhat monotonous manner), or because there is a stress cycle on the contact at lower stress levels (in a fatigue-related manner). When the bond is broken, the contact switches to a model similar to the "Cohesion" one. Bonds can be initialized (or reinitialized) to the "intact" status if the keyword "InitializeCZM" is written in the DYNAMIC file used as the starting point of a simulation. A given contact will be initialized as "intact" if the normal gap is lower that a certain value gap_{init}.

Keyword: '***CZMfatigue***'

Parameters:

k_{ini}	(intact normal stiffness)
P_{nlim}	(tensile cut-off stress)
gap_{nlim}	(tensile cut-off gap)
p_{nres}	(residual cohesion when the bond is broken)
nt_{ratio}	(ratio of all the tangential parameters to the normal ones)
P_{fat}	(stress from which fatigue-related damage is active)
$Rate_n$	(rate of fatigue damage in the normal direction)
$Rate_t$	(rate of fatigue damage in the tangential direction)
gap_{init}	(initial detection distance to set a contact as "intact")

III. Executing the code

1. Generating the data files

When the matlab data script (for example MELODYLoad Data_Example.m) has been created, it should be called from the master preprocessing script MELODY.m, and this script should be executed in matlab. It might take some time if the simulation involves the discretization and the pre-integration of many different bodies, but after some time the pre-processor will provide three text (.asc) files, namely: STATIC_DATA.asc, STATIC_CONTROL.asc, and DYNAMIC_00000.asc. Those are the three files that are needed in order to launch a simulation.

2. Running a simulation under Windows

Small simulations can be run under Windows. In that case, the three .asc files must be copied in a dedicated directory, as well as the following files (which are provided in the main package of the code):

-MELODY_2D.exe
-libgcc_s_seh-1.dll
-libgomp-1.dll
-libstdc++-6.dll
-libwinpthread-1.dll

A simulation can be launched directly by double-clicking on the executable file, or using the classical Windows command tool. In that case, after navigating to the proper directory, the syntax to execute the code is:

path>MELODY_2D.exe 0

The argument 0 means that the file DYNAMIC_00000.asc will be used as the starting point of the simulation. However, if a previous simulation is to be started again for some reason, it can be from any saved DYNAMIC_nnnnn.asc file. In such case, just replace the number "0" by the index of the desired starting file.

By default, the simulation will use all the CPUs of the computer, and this may slow down the execution of Windows or of any other program running at the same time. If one wants to assign a limited numbers of CPUs to MELODY, it can be done in the Windows command by using the following syntax (before executing the code, of course):

path>set OMP_NUM_THREADS=n

with n the number of desired threads.

3. Running a simulation on the LaMCoS cluster

If you are part of the LaMCoS, you do not need to compile MELODY, since a Linux compiled version is included in the package (if you are working on another cluster, you might need to recompile MELODY from the provided source files, and you can probably ignore this section).

Communication with the cluster is done by the ssh protocol. If you are on a desk computer in the lab, you can connect directly, but if you are on a laptop or outside of the lab, you will need first to establish a secured VPN link, for example using Cisco Any Connect. The necessary information for this connection is:

-Name of the vpn: vpn.insa-lyon.fr
-Username: sname (where s is the first letter of your surname and name is your name)
-Password: the one of your LaMCoS account

You can then establish the SSH connection, with two tools:

-SSH Secure Shell (for the command line)
-SSH Secure File Transfer (to send or receive files from the cluster)

Both programs are supposed to be already installed on your LaMCoS computer. You will be asked for the following information:

-Host Name: lamcoscalc
-User Name: sname (where s is the first letter of your surname and name is your name)
-Password: the one of your LaMCoS account
-Port Number: 22

When connected, you will need to access to your personal space in the "scratch" disc in SSH Secure File Transfer. To do that, go back to the root of the cluster, find the scratch directory, and find yourself in the list of available directories: this is your personal computation space. In there, create the directories you need for computation, and copy in the same directory the following files:

-MELODY_2D (the Linux executable compiled file)
-STATIC_DATA.asc
-STATIC_CONTROL.asc
-DYNAMIC_00000.asc
-Launcher_MELODY.sh

The file "Launcher_MELODY.sh" is the launcher of your computation, it defines the resources you are asking for, among other things. An illustrative file is provided in the MELODY package, which is a good starting point. You just need to replace:

-"MELODY_MySim" by the name of your simulation
-"ppn=20" by "ppn=n" where n is the number of threads you want
-"mem=4GB" by "mem=nGM" where n is the amount of memory you want
-"96" by the number of computation hours you need
-"surname.name@insa-lyon.fr" by your email address
-"./MELODY_2D 0" by "./MELODY_2D n" where n is the index of the starting DYNAMIC file

When this is done, you can turn back to SSH Secure Shell to run the simulation. To go to your personal scratch space, use the following syntax:

-*bash$* cd ../../../..
-*bash$* cd scratch/sname/*path*/

where *path* is the directory where you want to run a simulation. Then, you need to do two things:

-*bash$* chmod +x MELODY_2D
-*bash$* module load gcc/5.3.0

The first one only needs to be done once when you copy the executable MELODY_2D in a certain folder, the second one should be done each time you start a new session on SSH Secure Shell.

You are now ready to launch your simulation by typing:

-*bash$* qsub Launcher_MELODY.sh

This will assign a number to your job and place it in the queue of the simulations. The current state of the queue can be consulted using:

-*bash$* qstat

Each computation has a number, a name, a user, and a status: Q (queuing), R (running) or C (completed, either by success or failure). You can stop your simulation by typing:

-*bash$* qdel number

where number is the number of your simulation. Otherwise the simulation will stop when the computation is over, or when the walltime (specified in the launcher) is reached. Output files can then be collected from SSH Secure File Transfer for postprocessing.

There are a few interesting rules to know about the LaMCoS cluster:

-Each user can use 80 CPUs at a given time.
-Each user is limited to 2 simulations of 240 hours (which is the maximum allowable) and 10 simulations of 96 hours at a given time. Shorter simulations are also limited but the limit is never reached in practice.
-Our largest computer has 32 CPUs, but most of them have 20 (the best ones) or 8 (the older ones). Use the ones with 20 in priority.
-The more you ask in resources (CPUs, memory, walltime), the less priority you have in the queue.

4. Postprocessing

During execution (either under Windows or on a cluster), the code will write some files of several categories:

-DYNAMIC files, which contain the totality of the current state of the system, and can be used to launch again a new simulation

-GRAPHIC files, which can be open in Paraview

-Spying files, which are specified in the SPIES section of STATIC_CONTROL.asc

Postprocessing can be done on any of these files by the techniques chosen by the user. However, two tools are provided to automatize some operations:

-BATCHPRINT can write the graphic files in batch from the DYNAMIC files present in the directory. The syntax is (either in Windows or in a dedicated launcher on the cluster):

BATCHPRINT n_{start} n_{step} n_{end}

-BATCHEXTRACT can extract some new spies from the DYNAMIC files present in the directory. In that case, the desired spy should be specified first in the STATIC_CONTROL.asc file. The syntax is (either in Windows or in a dedicated launcher on the cluster):

BATCHEXTRACT n_{start} n_{step} n_{end}

Obviously, only data from the saved states (i.e. those written in the DYNAMIC files) can be extracted, so the period of the spy should be chosen adequately.

IV. Syntax of the data files

There are four main types of data files in MELODY:

-STATIC_DATA.asc contains all the preprocessed data generated by the preprocessor. It is usually a quite big file, which should not in principle be modified by the user.

-STATIC_CONTROL.asc contains all the data that the user can modify if he/she wants to change any parameter of the simulation.

-DYNAMIC_00000.asc (and following numbers) describes the complete state of the simulation at a certain time. It can thus be used to relaunch the simulation from this state, or to e*xtract relevant data.

-GRAPHIC_00000.vtk (and following numbers) is a graphic file that can be open in Paraview for visualizing a certain number of quantities in the system.

Hereafter, we only detail the content of STATIC_CONTROL and of the DYNAMIC files, since they allow the user to have a certain amount of control of the simulation without the need to launch again completely the matlab preprocessor.

1. STATIC_CONTROL file

Most of the data contained in this file (but not all of them) can be easily modified. Its syntax obeys to quite strict rules, and the general structure should not be altered or the file will not be readable any more by MELODY. More specifically, the order in which the data are given and the number of lines between them should be respected. The first main section of the file contains general data (in brackets, the meaning of the parameters is indicated, and the empty lines are noted *):

```
*
SIMULATION_NAME
[name of the simulation]
*
*
MATERIALS
[number of different materials]
*
[name of the first material]
[type of material]
[parameters of the material]
*
[name of the second material]
...
*
*
CONTACT_LAWS
[number of contact laws]
*
```

[first material of the first contact law]
[second material of the first contact law]
[type of the first contact law]
[Evolutive or Fixed contact]
[parameters of the firstcontact law]
*
[first material of the second contact law]
...
*
*
SOLVER
[type of solver]
[initial time] [initial time step] [endin time]
[solver parameters]
[DYNAMIC saving period] [GRAPHIC saving period] [contacts updating period]
*
*
PERIODIC_BOUNDARIES
[Xmin] [Xmax]
*
*
GRAVITY
[Along x] [Along y]
*
*
NUMBER_BODIES
[number of bodies]
*
*
MONITORING
[number of monitorings]
*
*
SPIES
[number of spies]
*
[name of the file of the first spy] [number of spied quatities] [spying period]
[first spied quantity]
[second spied quantity]
...
*
[name of the file of the second spy] [number of spied quatities] [spying period]
...
*
*
DEACTIVATION
[number of deactivations]
*
*

GRAPHIC
[0 or 1] [name of quantity 1]
...
[0 or 1] [name of quantity 27]
*

In the remainder of the file, all the bodies (both rigid and deformable) are described, with a similar syntax:

*
[RIGID or DEFORMABLE]
[number of the body]
[name of its material]
[type of body, i.e. simple or periodic]
[typical nodal distance] [proximity detection distance] [contact detection distance]
*
NODES
[number of nodes]
*
BORDERS
[number of borders]
*
[type of the first border, i.e. Closed, Simple, or Periodic] [type of interpolation on the border]
[number of nodes on the first border]
X [type of boundary condition] [possible parameters of the boundary condition]
[number of history points for the boundary condition, if it exists]
[time of the first history point] [prescribed value of the boundary condition at that time]
[time of the second history point] [prescribed value of the boundary condition at that time]
…
Y [type of boundary condition] [possible parameters of the boundary condition]
[number of history points for the boundary condition, if it exists]
[time of the first history point] [prescribed value of the boundary condition at that time]
[time of the second history point] [prescribed value of the boundary condition at that time]
…
*
[type of the second border, i.e. Closed, Simple, or Periodic] [type of interpolation on the border]
…
*
INTEGRATION
[number of Gauss points] [number of regions to distribute on different CPUs]
*
TRIANGULATION
[number of triangular facets in the body, for numerical integration and graphic output]
*

2. DYNAMIC files

All the dynamic files can be edited manually, but some of their data should not be modified because they point to STATIC_DATA or STATIC_CONTROL. The first section contains general data:

```
*
SIMULATION_NAME
[name of the simulation]
*
SOLVER
[type of solver]
[current dynamic number] [current iteration] [current time] [current graphic number]
[current time step]
[next saving time] [next graphic time] [next contact updating time]
*
```

Then comes an optional section with keywords that enable some functionalities at the beginning of the simulation. There may be any number of them. Here is the list of these keywords and of their meaning:

KILL_VELOCITY: all the velocities of all the degrees of freedom of the system are set to zeros in the initial state. This functionality is useful when one wants to dissipate rapidly kinetic energy, for example to accelerate a gravitational deposition process.

MONITOR_ENERGY: Enables monitoring of the system energy at every time step.

MONITOR_BOUNDARIES: Enables monitoring of the periodic bodies at every time step.

INITIALIZE_CZM: Restores all the contacts with a bond-braking law (e.g. BondedMohrCoulomb or CZMlinear) to the status “intact”. The internal damage variable is set to 0 for any detected contact of that kind. This keyword is mandatory in DYNAMIC_00000.asc when such laws are used, otherwise no bond will exist at all in the simulation.

UPDATE_MASS_MATRIX: Updates the mass matrix of all the bodies, in case of modification of their density in the list of materials. If this keyword is not activated, any change of density of the materials will be accounted for the code, but the mass-related Rayleigh damping will be left unmodified.

UPDATE_STIFFNESS_MATRIX: Updates the stiffness matrix of all the bodies, in case of modification of their mechanical parameters (or mechanical model) in the list of materials. If this keyword is not activated, any change of mechanical properties of the materials will be accounted for the code, but the stiffness-related Rayleigh damping will be left unmodified. This computation may be quite long, and should only be used if strictly necessary.

UPDATE_DAMPING_MATRIX: In case of use of any of the previous keywords, this one updates the mass matrix based on the previous changes in the mass and/or stiffness matrices.

NO_LOG: No text is printed in the command window during execution, and no log file is written.

NO_MONITORING: No monitoring is made during execution.

Finally all bodies are described and their current states are detailed. For each deformable body, the following data are available:

*
DEFORMABLE
[status: active or inactive]
[number of the body]
*
KINEMATICS
[number of nodes]
[number of the first node] [x-current position] [y-current position] [x-displacement] [y-displacement] [x-velocity] [y-velocity] [x-acceleration] [y-acceleration] [x-displacement DOF] [y-displacement DOF] [x-velocity DOF] [y-velocity DOF] [x-acceleration DOF] [y-acceleration DOF]
[number of the second node]…
…
*
FORCES
[number of the first node] [x-internal force] [y-internal force] [x-contact force] [y-contact force] [x-self contact force] [y-self contact force] [x-body force] [y-body force] [x-Dirichlet force] [y-Dirichlet force] [x-Neumann force] [y-Neumann force] [x-damping force] [y-damping force] [x-total force] [y-total force]
[number of the second node]…
…
*
WORKS
[cumulated internal work] [cumulated contact work] [cumulated body work] [cumulated Dirichlet work] [cumulated Neumann work] [cumulated damping work] [cumulated ALID work]
*
NEIGHBOURS
[number of proximities detected]
[number of the first detected body] [number of the detected border of that body] [periodicity]
[number of the second detected body]…
…
*
BORDERS
*
[number of the first border]
[number of border segments]
[length of the first border segment] [number of the node 0] [periodicity of the node 0]
[number of the node 1] [periodicity of the node 1] [number of the node 2] [periodicity of the node 2] [number of the node 3] [periodicity of the node 3]
[length of the second border segment]…
…
*
[number of the second border]
…
*
CONTACTS
[number of contacts]

[number of the slave border for the first contact] [number of the salve border node] [number of the slave node] [number of the master body] [number of the master border] [number of the periodicity shift] [number of the master segment] [normal gap] [tangential gap] [Xsi parameter] [x-component of the unit normal vector] [y-component of the unit normal vector] [x-component of the unit tangential vector] [y-component of the unit tangential vector] [length of the two half segments around the contact] [x-contact force] [y-contact force] [number of the master node 0] [contact shape function at the master node 0] [periodic shift of node 0] [number of the master node 1] [contact shape function at the master node 1] [periodic shift of node 1] [number of the master node 2] [contact shape function at the master node 2] [periodic shift of node 2] [number of the master node 3] [contact shape function at the master node 3] [periodic shift of node 3] [contact effective mass] [number of contact internal variables] *optional:* [contact internal variables]
[number of the slave border for the second contact]…
*
CONTACT_PRESSURES
*
[number of the first border]
[number of nodes in the first border]
[x-contact pressure for the first node] [y-contact pressure for the first node]
[x-contact pressure for the second node] [y-contact pressure for the second node]
…
*
[number of the second border]
…
*
BC_PRESSURES
*
[number of the first border]
[number of nodes in the first border]
[x-boundary pressure for the first node] [y-boundary pressure for the first node]
[x-boundary pressure for the second node] [y-boundary pressure for the second node]
…
*
[number of the second border]
…
*

For rigid bodies, the syntax is the same, except that the keyword RIGID is used instead of DEFORMABLE, and the sections KINEMATICS and FORCES are replaced by:

*
KINEMATICS
[x-position] [y-position] [theta-position]
[x-displacement] [y-displacement] [theta-displacement]
[x-velocity] [y-velocity] [theta-velocity]
*
FORCES
[x-contact force] [y-contact force] [theta-contact force]
[x-body force] [y-body force] [theta-body force]
[x-Dirichlet force] [y-Dirichlet force] [theta-Dirichlet force]

[x-Neumann force] [y-Neumann force] [theta-Neumann force]
[x-damping force] [y-damping force] [theta-damping force]
[x-total force] [y-total force] [theta-total force]
*

V. General remarks

-There are no units implemented natively in the code, it is completely dimensionless. Proper scaling makes it possible to apply it to any system of units, provided the quantities introduced in the input files are dimensionally consistent.

-All lists (bodies, nodes, borders, materials, contacts, etc.) start at 0 in the main code, which is sometimes in contradiction with the matlab numbering. Hence, to avoid mistakes, one should keep in mind that the element 1 of (for example) the matlab variable **Contours** corresponds to the body 0.

VI. Troubleshooting

This section will be developed little by little based on feedbacks from users. Do not hesitate to share your difficulties, in order for me to provide easy tricks to solve them quickly.

VII. Release notes

-24th of March 2019 – V. 3.83.

Corrected a bug in the "BondedMohrCoulomb" contact law, which now uses the contact element length which is stored as a contact variable. Also, the "CZMfatigue" contact law was slightly modified: the stress level from which fatigue may occur is now directly fed as a law parameter (instead of having it defined by a ration of the Plim parameter).

-29th of May 2019 – V. 3.87.

Added an initialization parameter for the damageable contac laws (i.e. "BondedMohrCoulomb", "CZMlinear", and CZMfatigue"). It is a detection normal gap, below which a given contact will be initialized as "intact" if the keyword "INITIALIZE_CZM" is present in the called DYNAMIC file. Otherwise, the contact will be initialized with the "broken" status.

The preprocessor was slightly modified, regarding the "Mesh_Ratio" variable for the "Polygon" body type. See related section in this user guide.