DEMPLA and OVAL: programs for analyzing particle assemblies and for simulating wave propagation and liquefaction with the Discrete Element Method

Matthew R. Kuhn*, kuhn@up.edu

May 23, 2021
DEMPLA version 0.2.1

1 Introduction

DEMPLA means "Discrete Element Method for Propagation and Liquefaction Analysis." Two programs are included as two modes of the same DEMPLA code. The two modes incorporate the older code OVAL, which serves as the underlying back-engine for Discrete Element Method (DEM) simulations and of individual particle assemblies and of the multiple assemblies of a one-dimensional soil column. In the first mode, the program OVAL (which is contained within the larger DEMPLA code) performs DEM analysis of a single particle assembly. In the second mode, DEMPLA uses multiple DEM assemblies to analyze one-dimensional wave propagation within a soil column. The DEM is a method for simulating the motions and interactions of the individual particles in a granular material while the entire assembly is being deformed (1979). Perhaps most important, the code allows extracting global averages of such quantities as stress and fabric for the entire assembly as well as the micro-level quantities, such as particle movements and contact forces. In the first mode, the program is primarily intended for analyzing "dense" assemblies (as opposed to diffuse assemblies) that are roughly rectangular in shape, such as in so-call *element tests*. In the first mode, the code can also be used to create assemblies by compacting a diffuse assembly into a denser state. The program's capabilities in this mode are summarized in Section **??**. In the first mode, the code runs the DEM code with a single computing thread.

*Dept. of Civil Engineering, Donald P. Shiley School of Engineering, University of Portland, 5000 N. Willamette Blvd., Portland, OR 97203 USA, kuhn@up.edu.

In the first mode, the program handles both two- and three-dimensional simulations with particles that are either circles (2D), ellipses (2D), ovals (2D), spheres (3D), "nobby" clusters of circles (2D), "bumpy" clusters of spheres (3D), a special non-spherical "ovoid" particle (3D).

In the second mode, the code uses 3D DEM assemblies as the representative volume elements (RVEs) within a one-dimensional soil column to analyze wave propagation and liquefaction in the column. In this mode, the code is a multi-phase, multi-scale rational method for modeling and predicting free-field wave propagation and for modeling the weakening and liquefaction of near-surface soils. The one-dimensional time-domain model of a soil column uses the discrete element method (DEM) to track stress and strain within a series of representative volume elements (RVEs), driven by seismic rock displacements at the column base. The RVE interaction are accomplished with a time-stepping finite-difference algorithm. The method applies Darcy's principle to resolve the momentum transfer between a soil's solid matrix and its interstitial pore fluid. Different algorithms are described for the dynamic period of seismic shaking and for the post-shaking consolidation period. The method can analyze numerous conditions and phenomena, including site-specific amplification, down-slope movement of sloping ground, dissolution or cavitation of air in the pore fluid, and drainage that is concurrent with shaking. Several refinements of the DEM are described for realistically simulating soil behavior and for solving a range of propagation and liquefaction problems, including the poromechanic stiffness of the pore fluid and the pressure-dependent drained stiffness of the grain matrix. The model is applied to the conditions of four sets of well-documented centrifuge studies. The verification results are favorable and highlight the importance of the pore fluid conditions, such as the amount of dissolved air within the pore water. In the second mode, the several DEM algorithm can be run with multiple computing threads to analyze the many DEM assemblies (RVEs) in parallelized fashion.

In the second mode, other than being restricted to one dimension, the method is quite general and allows one to model and understand diverse conditions and phenomena: (1) three-dimensional motions of rock and soil, propagating as both p-waves and s-waves; (2) nearly arbitrary stress and strain paths during ground shaking; (3) sloping ground surface with down-slope movement; (4) multiple stratigraphic soil layers; (5) sub-surface water table or complete submergence of the site; (6) sloping water table with down-dip seepage forces; (7) onset and depth of liquefaction; (8) saturated soil, dry soil, or quasi-saturated soil with entrained air at a specified saturation; (9) dissolution or cavitation of entrained air; (10) effect of saturation on wave propagation and liquefaction; (11) effect of drainage that is concurrent with shaking; (12) pore fluid with a specified viscosity; (13) site-

specific amplification of surface accelerations relative to those of the rock base; (14) pressure-dependence of wave speed and the slowing of waves as they approach the surface; (15) dilation and the coupling of s-waves and p-waves; (16) voids redistribution and development of a water film beneath less permeable layers; (17) softening of soil during shaking, with a slowing of wave speeds and shifting of frequency content due to build-up of pore fluid pressures; (18) post-shaking consolidation and settlement; (19) post-consolidation reshaking and post-triggering behavior; and (20) relative rise of the water table during and after shaking. The authors has not fully investigated most of the possibilities.

The code is freely available on GitHub. If you find opportunities to improve the code or if you find errors, please let me know or fork the code and augment or correct the code yourself. The author compiles the Fortran code on BSD Unix and Gnu Linux systems, although the code can likely be compiled on windows systems, too.

Besides the code, the GitHub repository also contains sample assemblies, example problems, and Octave code for analyzing results.

2 Contents

3 Capabilities and limitations

3.1 Capabilities and limitations of the code

The its current version the code has the following capabilities and limitations:

1. In the first mode it can perform discrete element analysis on assemblies containing any one of the following particle types: circles (2D), ellipses (2D), ovals (2D), spheres (3D), "nobby" clusters of circles (2D), "bumpy" clusters of spheres (3D), and non-spherical "ovoids" (3D). In the second mode, only the 3D shapes can be used.

2. At present, it works best with flat periodic boundaries on all sides of a two- or three-dimensional assembly. For two-dimensional assemblies (in the first mode), it can also use flexible boundaries; with both two- or three-dimensional assemblies, it can use rigid boundaries. It can only analyze assemblies with parallel boundaries (i.e. assemblies of 2D parallelogram and 3D parallelepiped shapes).

3. The code is not intended for boundaries of arbitrary shape or for arbitrary boundary conditions.

4. The code is intended for dense (jammed) particle assemblies that are slowly deformed at quasi-static rates. Although the code can also model rapid flows and diffuse assemblies, this purpose was not a primary concern in the code's design.

5. It can create output files of both macro-results and micro-results for the DEM assemblies. The macro-result files include information on stress and deformations. The micro-result files give information on particle positions, particle orientations, contact forces, and system topology.

6. In the second mode, the program can create the same output files for each of the DEM assemblies in the soil column, as well as average assembly values of both the grain matrix and pore fluid for all DEM assemblies along the height of the soil column.

7. It includes a simple contact force mechanism consisting of linear springs (both normal and tangential) and a frictional slider. The normal and tangential spring constants can be specified independently (Sections 8.1.15 and 8.1.16). The frictional slider can be "turned off" (Sections 8.1.17 and 8.1.18).

8. It also includes more advanced contact models: Hertz-Mindlin models of sphere-sphere contacts, cone-cone contacts, and contacts of intermediate shape (surfaces of revolution of power-form contours). With these models, the grains' shear modulus, Poisson ratio, and friction coefficient must be specified.

9. The code can analyze assemblies with a poromechanic description of the pore fluid. As such, the code directly computes the pore fluid pressure as well as the inter-granular (effective) stress, and it can adjust the inflow of pore fluid as well as the deformation of the grain matrix.

10. Available poromechanic models include dry assemblies, saturated assemblies, and quasi-saturated assemblies in which gas is presents as entrained bubble that do not form a continuous network within the pore space separate from the pore liquid.

11. The following pore fluid characteristics can be specified: saturation, bubble size or number density, gas solubility, surface tension, liquid bulk modulus, etc.

12. It includes the option of a modified DEM algorithm for self-regulating and maintaining the quasi-static nature of a simulation (Sections 8.1.2, 10.3.5, 10.3.15, and 14).

13. It includes the following types of damping: translational mass (global) damping, rotational mass (global) damping, contact damping (refer to 1979), and the damping method of Potyondy and Cundall. Each may be independently specified or "turned off" (Sections 8.1.21–8.1.23).

14. It includes a robust servo-control algorithm for controlling the boundary stresses. The deformation (or stress) path is supplied by the user in a series of steps, which specify in a component-by-component manner either the deformation rate tensor or stress rate tensor. The specified stress or deformation rate components may be mixed. See Section 8.2.

15. With stress-control, either the effective stresses or total stresses can be controlled in arbitrary combinations.

16. When a pore fluid is modeled, either the fluid pressure or the fluid inflow can be controlled. This capability is necessary for analysis wave propagation and liquefaction in the second mode.

17. It can create binary "restart" files that allow a new simulation to begin at the exact ending condition of a previous simulation. These restart files include all of the position, velocity, and contact information that

allow the new run to begin where the previous run had stopped. See Sections 8.1.7, 8.1.8, and 8.2.7.

18. It can assign initial random velocities to the particles in the assembly (Sections 8.1.24 and 8.1.25).

19. As an option, it can prevent particle rotations, particle motions, or both.

3.2 Things to do

Please help write this section.

4 Availability and installation

The code is written in a rather inelegant dialect of Fortran 77. The entire program and other files can be downloaded from the GitHub repository, including the source code, this documentation, sample assemblies, and example problems. Source code is freely available under terms of the open source GNU General Public License (GPL), version 2.

The program can be run in two modes. Running DEMPLA requires several input components:

1. In the first mode (stand-along simulation of a single particle assembly), the following input files are required:

   - The compiled executable file of DEMPLA, as described below.
   - A RunFile that specifies the contact model, the poromechanic model (if any), the general conditions of the simulation, and the sequence of stress-strain control steps to be performed by the simulation.
   - A StartFile that gives the size of the assembly, the numbers of particles in the assembly, and the locations, sizes and orientations of all particles in the assembly.

2. In the second mode (wave propagation and liquefaction analysis of a one-dimensional soil column), several input files are required. Many of these files must be arranged within a particular directory (folder) structures, described later.

   - The compiled executable file of DEMPLA, as described below. The source code should be compile with the OpenMP parallelization library, as described below. Shell variables should also be set to permit multi-thread running of DEMPLA, as described below.

- A GFile that specifies the number of DEM assemblies in the soil column, the height of the column, the number of stratigraphic soil layers, water depth, and other general characteristics of the simulation.
- One LFile for each of the stratigraphic soil layers, giving the RunFile and StartFile for setting up and running the DEM algorithm for the stratigraphic layer.
- A RunFile for each stratigraphic layer that specifies the contact model, the poromechanic model (if any), the general conditions of the simulation, and the sequence of stress-strain control steps to be performed by the simulation.
- A StartFile for each stratigraphic layer that gives the size of the assembly, the number of particles in the assembly, and the locations, sizes and orientations of all particles in the assembly.
- A MotionFile that gives the displacement sequence at the base of the soil column.

4.1 Compiling Dempla on Linux systems

The following steps are required to create a compiled executable DEMPLA file and to set up multi-thread running of the code (second mode, only). If you are only running the program in first mode (for a single assembly), only steps 1 and 2 are required, and steps 3 through 6 can be ignored. The author has compiled the code with both the `gfortran` and `ifort` compilers. If you compile with other compilers, please augment the descriptions below with your own experience.

1. Download or pull the following three files from the GitHub repository:

   - `dempla-X.X.XX.f`, the latest version of the main DEMPLA source code.
   - `common-dempla-X.X.XX.f`, the corresponding file of the common blocks used throughout the DEMPLA code.
   - `param-dempla-X.X.XX.f`, the corresponding file of parameters that sets the sizes of arrays in the DEMPLA code.

   These three files should be placed in the same build folder.

2. DEMPLA can be parallelized by compiling with the proper OpenMP flag, which for the gcc `gfortran` and Intel `ifort` compilers are as follows:

   `gfortran`: `-fopenmp`
   `ifort`: `-qopenmp`

   To compile with optimization in the Linux shell:

```
gfortran -std=gnu -O3 -mtune=native \
                  -mcmodel=large \
                  -fopenmp \
                  -o dempla.exe
                  dempla-X.X.XX.f
```

or

```
ifort -ipo -O3 -no-prec-div -fp-model fast=2 -xHost
      -fpconstant \
      -mcmodel=large \
      -qopenmp -heap-arrays \
      -o dempla.exe
      dempla-X.X.XX.f
```

The options `-std=gnu -O3 -mtune=native` and `-ipo -O3 -no-prec-div -fp-model fast=2 -xHost` produce optimized code. The options `-fltconsistency -fpconstant` improve floating point precision. Option `-mcmodel=large` is necessary for large compiled arrays (as might be needed when there are millions of particles or when there are thousands of particles but with the contact history parameter `mlistJ` being exceptionally large). The options `-fopenmp` and `-qopenmp` specifiy compiling with the OpenMP library. Option `-heap-arrays` places temporary arrays that are used within subroutiens (i.e., arrays that are treated as private and take new values with each threaded instance) onto the heap instead of the stack. Finally, option `-o dempla.exe` specifies the name of the executable file.

To preclude parallelization and always run the code as a single thread (for example, with the first mode), simply recompile dempla without the `-fopenmp` or `-qopenmp` flags.

Before compiling DEMPLA, you will need to consider the values of various parameters that specify the sizes of arrays. These parameters are described and given in the file `param-dempla-X.X.XX.f`. The most important parameters are `mp` (the maximum number of particles in the DEM assemblies) and `mrve` (the maximum number of DEM assemblies (RVEs) in the soil column, when running the second mode). For example, if you are analyzing a soil column and wish to have 30 DEM assemblies in the column, you will need to set `mrve=30` in the file `param-dempla-X.X.XX.f` before compiling.

Because the compile commands are cumbersome, you might consider creating an `alias` within the `.bashrc` configuration file.

3. In order for parallelization to work within a Linux shell (i.e., a terminal window), you should specify number of threads to be used during

runtime inside of the shell. The number of threads is given by the shell environment parameter `OMP_NUM_THREADS`. Within a Linux bash shell:

```
export OMP_NUM_THREADS=4
```

which would enable parallelized runs with 4 threads. The authors typically uses 10 threads on a 28-thread Xeon processor. This step must be done within *every* shell (terminal) within which the program is run, or it can be included in the `.bashrc` file or in another shell initialization script that can be sourced when needed. Note that if no value of `OMP_NUM_THREADS` is set, then the code will possibly run with all possible threads on the CPU, which is usually not desirable.

You can query the current value of the environment parameter with this command:

```
echo $OMP_NUM_THREADS
```

To reset the `OMP_NUM_THREADS` parameter, within a Linux bash shell:

```
unset OMP_NUM_THREADS
```

To query the number of threads available on the computer's hardware:

```
lscpu | grep -E '^Thread|^Core|^Socket|^CPU\('
```

See this guide for servers with multiple sockets, as with Xeon and EPYC processors:
https://software.intel.com/en-us/forums/intel-moderncode-for-parallel-architectures/topic/739878.

Instead of setting the parameter `OMP_NUM_THREADS` in the shell, the parameter can also be assigned at runtime. For example, if the executable code is `dempla.exe`, the following can be run in the bash shell:

```
OMP_NUM_THREADS=4 ./dempla.exe
```

when 4 threads are desired.

4. For parallelized code, you might need to increase the stack size to accommodate the multiple threads. For example,

   ```
   export OMP_STACKSIZE=80m
   ```

   The default size of `OMP_STACKSIZE=80m` is usually small, 1m or 2m, and can lead to segmentation fault errors during runtime. You might need to try different sizes, until an adequate stack size is found, but without exhausting the machine memory. See this explanation:
   https://stackoverflow.com/ questions/13264274/ why-segmentation-fault-is-happening-in-this-openmp-code

   The parameter can also be assigned at runtime. In the bash shell:

   ```
   OMP_NUM_THREADS=4 OMP_STACKSIZE=80m ./dempla.exe
   ```

5. One must provide sufficient memory for parallel execution. The `gfortran`

and `ifort` compilers create executable programs that use stack memory instead of static memory. The limit on stack memory size will need to be increased. With Linux, use the following shell command:

```
ulimit -s unlimited
```

This command must be entered in any shell that will run the executable program, or it can be included in the .bashrc or other shell initialization script. Without this command, you will likely receive a segmentation fault error.

6. Finally, because of the many niggling steps that are required to prepare a Linux shell to run parallelized DEMPLA, you might consider placing all of these commands in a single Linux bash script file that can be sourced within a terminal. I use the following file `dempla.bsh` for setting up the use of DEMPLA in a bash shell:

   ```
   #!/bin/bash
   #
   ulimit -s unlimited
   export OMP_STACKSIZE=80m
   export OMP_NUM_THREADS=10
   ```

   but you will want to customize your own script and place it somewhere within your shell's `$PATH`.

   To run the script in shell, it must be sourced within the shell:

   ```
   source dempla.bsh
   ```

4.2 macOS systems

Please help to write this section.

4.3 Windows systems

Please help to write this section.

5 Helpful/essential utilities

Please help to write this section.

6 Running Dempla and Oval

DEMPLA is *not* yet an interactive program with a graphical user interface. At present, it runs only in a batch mode.

You would normally run DEMPLA (in either of the two modes) from within a terminal (e.g. an Linux xterm window, the MacOS terminal utility, or DOS console) with the following command at the system prompt:

```
<path><dempla_executable_name>
```

where `<path>` is the path to your executable file with its `<dempla_executable_name>` (e.g. `dempla.exe` or whatever name or symbolic link is given to the file). In the GitHub filesystem, a Linux executable file '`dempla.exe`' is located in the `bin/` directory. Instead of explicitly providing the `<path>`, you may wish to create a link (shortcut) to the `dempla` executable file (say, `dempla.exe`), place the file it in a directory in your system's search `$PATH`, or modify your system's search `$PATH` to include the directory that contains the `dempla` executable file.

After starting the `dempa` command, you will be queried for the run mode:

```
Mode: single-assembly (1) or wave propagation (2):
```

After entering the mode, you will be queried for various names and information, which will depend on the mode:

- In the first mode (`imode=1`, single-assembly mode), you will be queried for the names of two files,

  ```
  Name of the RunFile:

  Name of the StartFile:
  ```

  and the simulation will proceed to run. We will henceforth refer to these two files with the generic names "RunFile" and "StartFile", which are described in Sections 8 and 9. These two files must be created before running DEMPLA. The file names of the RunFile and StartFile must conform to Unix conventions (no spaces, asterisks, question marks, etc.) The RunFile must reside within the directory from which the DEMPLA is run. The StartFile can be located in another directory, in which case, you must enter the full path of the StartFile.

  The output will normally be written to a set of files, whose names will contain the core RunFile name. The various output files are listed in Table 1 and will be described later in the documentation. In the first mode, the output files are all placed within the directory in which DEMPLA is run.

  As an example, the output file `A<RunFile>.txt` will be created, noting that the name of this output file contains the core RunFile name. The file is created in the same director from which the DEMPLA program is run. This "`A`" file is a text file that can be imported into a spreadsheet, such as Excel or Gnumeric to view the average stresses and deforma-

tions of the assembly (note, *imported*, not opened). The file can also be read into Octave or Matlab with a utility function.

Before describing the detailed contents of the RunFile and the StartFile, you should know that the RunFile is an ASCII (text) file that describes how the simulation is to be run: boundary deformation rates, boundary stress rates, friction coefficients, spring constants, etc. (see Section 8). A StartFile gives the number of particles, particle type, and the initial particle arrangement (sizes, positions, etc., see Section 9). A StartFile may be of either ASCII or binary type (Table 1).

- Before you can run DEMPLA in the second mode (i.e. to simulate wave propagation), you must create a special directory structure that will hold the input files and output files. This directory structure must be created inside of the directory in which DEMPLA is being run. Three of these directories and their contents are as follows:

  – A directory named `MotionInput`. This directory contain the input motion file, which describe the base (rock) motion for the simulation. A MotionFile is a text file, having the format described in Section **??**. The directory `MotionInput` can either be a directory with that name or a symbolic link (with the name `MotionInput`) targeting a directory that contains the MotionFile.

  – A directory named `RunFiles` that contains all of the RunFiles that are used in the simulation. Each stratigraphic layer will have a RunFile. Each RunFile should reside inside the folder `RunFiles`. A RunFile is a text file with contents and format described in Section 8. The RunFile describes the conditions under which the assemblies within the layer will be initialized and run: contact models, friction coefficients, spring constants, poromechanic fluid properties, etc. The names of RunFiles must conform to Unix conventions. The directory `RunFiles` can either be a directory with that name or a symbolic link (with the name `RunFiles`) targeting a directory that contains the RunFiles.

  – A directory named `StartFiles` that contains all of the StartFiles that are used in the simulation. A StartFile is a text file with contents and format described in Section **??**. Each StartFile is a text file (a DFile) that gives information about the particle assembly at the start of a simulation: assembly size, number of particles, and particle sizes, locations, and orientations. Each stratigraphic layer will have a StartFile. Each StartFile should reside inside the folder `StartFiles`. The names of StartFiles must conform to Unix conventions. The directory `StartFiles` can

either be a directory with that name or a symbolic link (with the name `StartFiles`) targeting a directory that contains the StartFiles.

The above three directories have have general role, since many simulations can reference these directories and can be run inside of the directory that contains them (i.e., the directory in which DEMPLA is being run). Besides the three directories, you must also create a directory that will contain input information and output for simulations on a particular soil column, noting that different input motions can be applied to the same soil column. A directory must be created with the BaseName, which must conform with the Unix naming conventions. The BaseName directory must also contain the following contents:

- The BaseName directory must contain three sub-directories with the following names: `02_LayerSetup`, `03_RVESetup`, and `04_RunOutput`. These three sub-directories will be used to hold pre-processing files and the output files that are created by the simulations. Various output files will be placed inside the sub-directory `04_RunOutput`, and these output files will be given names that contain the BaseName and the name of the MotionFile, so that the output sub-directory can contain output from multiple simulations, each with a different input motion. The various output files contain pore fluid pressures, soil displacements, effective stresses, etc. The range, format, and content of the various output files are described in Section **??**.
- The BaseName directory must contain a file that gives general information about the soil column: the number of DEM assemblies (RVEs) along the height, the number of stratigraphic layers, the column height, water depth, etc. This ColumnFile must have the name `GBaseName<suffix>`, noting the "`G`" prefix. As such, we refer to this file with the interchangeable references ColumnFile and GFile. The format and content of GFiles are described in Section **??**. The name of the GFile can include an optional prefix (several simulations can be run with the same BaseName and input motion, but can differ by the pore fluid saturation, and these variations can be distinguished with the suffix. See Section **??**).
- The BaseName directory must also contain files that give information about all of the stratigraphic soil layers that are within the soil column. One LayerFile must be created for each of the soil layers (note that the water table can be within the thickness of a soil layer, but only one LayerFile is required for the layer). The

LayerFile must have the name L<XXXX>_BaseName. The name begins with letter "L", followed by a 4-digit number (beginning with 0001 for the top soil layer, and increasing with 0002, 0003, etc. to the bottom layer). The number is followed by an underscore "_" and the BaseName.

We will refer to this file with the interchangeable references LayerFile and LFile. The format and content of LFiles are described in Section ??. The LFile gives the thickness of the layer, void ratio, etc. Most important, the LFile also gives the names of the RunFile and StartFile for the layer.

The RunFile for a layer provides general information about the DEM assembly that represents the particular stratigraphic layer. The RunFilemust be contained in the RunFiles directory (i.e., one of the three general sub-directories, described earlier). A RunFile is a text file with contents and format described in Section 8. The RunFile describes the conditions under which the assemblies within the layer will be initialized and run: contact models, friction coefficients, spring constants, poromechanic fluid properties, etc. The names of RunFiles must conform to Unix conventions.

The StartFile for a layer gives information about the size of the DEM assembly that represents the soil layer, along with the shapes, locations, sizes, and orientations of all of the particles in the assembly. A StartFile is a text file with contents and format described in Section ??. Each stratigraphic layer will have a StartFile. Each StartFile should reside inside the folder StartFiles. The names of StartFiles must conform to Unix conventions.

To run DEMPLA in the second mode (multi-assembly wave propagation), start the program and enter 2 as the mode. You will be queried for the following three names,

```
BaseName of the simulation (main folder name):

Suffix of variation (if none, press return):

MotionFile (name of rock motion file):
```

and the simulation will proceed to run.

File name	Type	Function	Sections
A<RunFile>.txt	text	macro-results for spreadsheets	10.1
B<RunFile>	text	macro-results for text editors	10.2
C<RunFile>	binary	restart StartFile	8.1.7
C?<RunFile>	binary	restart StartFile	8.2.7
D<RunFile>	text	StartFile	9
F[1-4]?<RunFile>	text	micro-results for post-analyses	10.5
? – a letter that corresponds to the deformation-stress path in which the file was created (Section 8.2.7 and Section **??**)			

Table 1: OVAL output files

7 Boundary Types

OVAL is primarily intended for element studies of using rectangular (2D) and box (3D) assemblies of particles. During a simulation, the boundaries (sides) are moved to produce prescribed rates of strain or rates of stress, as described in Section 8.2. The boundaries themselves can be of several types.

7.1 Periodic boundaries

The default boundaries are periodic. These are the easiest boundaries to use, and they can be used with either dense or sparse assemblies. Moreover, some of the other types of boundaries are created by starting with an assembly having periodic boundaries and then replacing the periodic boundaries with the another boundary type.

7.2 Tight-fitting particle boundaries

This type of boundary can only be created with 2D assemblies, and it is created by beginning with a non-sparse (at least, moderately dense) assembly having periodic boundaries. The process of creating a tight-fitting particle boundary involves finding the particle graph of the assembly (i.e., finding the topological arrangement of the contacts) and then identifying the string of contacting particles that surround the assembly. These particles become the boundary particles, which will fit tightly against (i.e., will be in contact with) the interior particles. After the periodic boundaries are "broken" and replaced with tight-fitting boundaries, the boundary particles will not likely be in equilibrium, so a period of several hundred time steps should be included to allow the assembly to equilibrate with its new boundaries. Once periodic boundaries are replaced with tight-fitting particle boundaries, the periodic boundaries can not be retrieved. Tight-fitting boundaries can be placed on the left and right sides (with periodic boundaries remaining

top and bottom), on the the top and bottom (with periodic boundaries remaining left and right), or on all four sides of the assembly. The intended combination of boundaries is specified with the `iflexc` input variable (Section 8.2.8).

Several types of stress or strain control are available with tight-fitting boundaries:

- Stress control (`iflexc = x1, 1x,` or `11`). The stress (actually, the stress rate) can be controlled with the `icontr=1` and `defrat` at the desired rate (Sections 8.2.1 and 8.2.2). For example, if tight-fitting boundaries are created on the left and right sides, the stress σ_{11} is applied against the two sides, and the rate of this stress can be controlled. In this same example, the other stress components (σ_{12}, σ_{21}, and σ_{22}) are also applied on the left and right sides, but only their original (not current) values are applied (those stresses present when the tight-fitting boundary was created). This approach prevents "hydrofracturing" of the side boundaries if the assembly is being compressed vertically. Stress-controlled tight-fitting boundaries approximate the membrane-type conditions that are commonly used in soil testing. The boundary stress is applied to imaginary boundary element: the branch vectors that connect the centers of the boundary particles.

- Displacement control with free rotation (`iflexc = x2, 2x,` or `22`). The particles along a tight-boundary are constrained to translate at a rate in accord with the prescribed strain rates (Sections 8.2.1 and 8.2.2). The boundary particles are allowed to rotate. Boundary forces are applied at the centers of the boundary particles.

- Displacement control with free rotation (`iflexc = x3, 3x,` or `33`). The particles along a tight-boundary are constrained to translate at a rate in accord with the prescribed strain rates (Sections 8.2.1 and 8.2.2). The boundary particles constrained to rotate in accord with the prescribed rotation rate (the Eulerian rate that corresponds to $\frac{1}{2}F_{12}$ in Section 8.2.1). Boundary forces are applied at the centers of the boundary particles.

- Displacement control with friction and free rotation (`iflexc = x4, 4x,` or `44`). Suppose that tight-fitting boundaries have been created on the left and right sides, and periodic boundaries remain on the top and bottom (`iflexc = 40`). With this type of control, particles along the left and right sides are constrained to translate horizontally at a rate in accord with the prescribed horizontal strain rates F_{11} and F_{12} (Sections 8.2.1 and 8.2.2). The side particles are free to translate vertically, but only if they overcome the side friction coefficient prescribed

by `frictw` (Section 8.1.18). The boundary particles are allowed to rotate. Boundary forces are applied at the centers of the boundary particles.

- Displacement control with friction and free rotation (`iflexc = x5, 5x,` or `55`). Suppose that tight-fitting boundaries have been created on the left and right sides, and periodic boundaries remain on the top and bottom (`iflexc = 40`). With this type of control, particles along the left and right sides are constrained to translate horizontally at a rate in accord with the prescribed horizontal strain rates F_{11} and F_{12} (Sections 8.2.1 and 8.2.2). The side particles are free to translate vertically, but only if they overcome the side friction coefficient prescribed by `frictw` (Section 8.1.18). The boundary particles constrained to rotate in accord with the prescribed rotation rate (the Eulerian rate that corresponds to $\frac{1}{2}F_{12}$ in Section 8.2.1). Boundary forces are applied at the centers of the boundary particles.

7.3 Rigid-flat boundaries

This type of boundary can be created with either 2D or 3D assemblies. The boundary is produced by giving an input value for `iflexc` of `9`, `90`, or `99` in the first line of the deformation-stress path section of a RunFile (see Section 8.2.8). A pair of rigid-flat boundaries (one each on opposite sides of the assembly) can coexist with periodic boundaries on the other sides. The size of the assembly (the box dimensions) are input with the dimensions `xcell` (Sections 9.1.2 and 9.1.3). Unlike with tight-fitting boundaries (Section 7.2), particles interact with rigid-flat boundaries at the particle surfaces, instead of at the particle centers. Stresses and strains can be controlled with rigid-flat boundaries, just as with other types of boundaries (Sections 8.2.1–8.2.5).

7.4 External-particle boundaries

These boundaries are created by surrounding the assembly with a set of external particles, which confine the interior particles. This type of boundary can be created with either 2D or 3D assemblies. The boundary is produced by giving an input value of greater than one to the integer `nplatn` (Section 8.1.13). For example, if `nplatn=4`, then you will be queried to give the names of four files that provide information about each of the four sets of boundary particles—their positions, radii, etc.—as described below. Stresses and strains can be controlled with rigid-flat boundaries, just as with other types of boundaries (Sections 8.2.1–8.2.5). Note that the boundary particles can only be circles (2D) or spheres (3D).

A file that provides information about a set of boundary particles contains four lines that give general information about the particles followed lines that provide information on each particle. The content of each line is described in the following subsections (Fortran free format is used, with integer or double precision type corresponding to the leading letter of the variable name).

7.4.1 `ipvers`

Set this value to 1. It gives a version number for the file, in the event that future changes are made to the format of these files.

7.4.2 `ixfix(1),ixfix(2),ixfix(3),ithfix(1),ithfix(2),ithfix(3)`

The manner in which the boundary particles are constrained in their motions. Values of either `0` or `1` (unconstrained or constrained, respectively) are assigned to the three directions of translation, (`ixfix`), and the three directions of rotation, (`ithfix`). Note that when translation is constrained in a particular direction, then the boundary particle move in accord with the prescribed deformation rate F_{ij} (Section 8.2.1).

7.4.3 `idirec`

The "direction" of the boundary. For example, if a set of boundary particles are one the left (i.e., x_1) side of a 2D assembly, then `idirec` is set to 1. If a set of boundary particles are one the top or bottom of a 2D assembly, then `idirec` is set to 2 (i.e., x_2). This feature is necessary to enable the control of stress within these boundaries.

7.4.4 `nplt`

The number of particles in the boundary file.

7.4.5 `rad,xp(1),xp(2),xp(3)`

The radius and position of a particle center, with one particle per line of input.

8 RunFiles for Oval

This ASCII text file is a formatted input file, which means that the input information must be placed within certain rows and columns (or column ranges). Sample RunFiles can be downloaded from the web site, and an example of a RunFile is shown in Fig. 1. This same file can be found at the

```
Prototype RunFile for the DEM program Oval
       2         : algori   | the algorithm for advancing the particle positions (1 or 2)
       1         : ivers    | whether to include additional lines in this RunFile
       0         : ncownt   | fequency for updating non-periodic boundaries (0)
       0         : iout(2)  | output files with avg. def. and gradients in layers (0 or 1)
       0         : iout(3)  | output files with avg. stresses within layers (0 or 1)
       1         : istart   | type of file defining the initial configuration (1, 2, or 3)
       0         : iend     | type of file to be created at end of the run (0, 1, 2, or 3)
       0         : idef     | reference configuration for reporting deformations (0)
     200         : iupdtm   | max. no. of time steps between linked-list updates
       0         : icirct   | compute and regularly update the particle graph (0 or 1)
       0         : imodel   | model for contact force
       0         : nplatn   | number of additional files with boundary particles
       8         : nloop1   | minimum number of iteration loops when algori=2
    1.00         : kn       | normal stiffness (force/indentation) or G (shear modulus)
    1.00         : kratio   | ratio tangential/normal stiffnesses or Poisson ratio
    0.50         : frict    | coefficient of friction at particle contacts
    0.50         : frictw   | coefficient of friction between particles and wall
    0.           : rho      | the mass density of the particle material
    0.400        : sep      | threshold separation during the near-neighbor searches
    0.05         : pcrit(1) | viscosity coefficient for translational body damping
    0.05         : pcrit(2) | viscosity coefficient for rotational body damping
    0.           : pcrit(3) | viscosity coefficient for contact damping
    0.64         : xseed    | seed for assigning random initial velocities (when motion=1)
    0.           : rmsvel   | rms initial velocity (when motion=1)
    0.           : pdif     | parameter for reducing Jager memory demand (when imodel=6)
    0.           : tmax     | maximum time for the simulation run
    0.           : A_1      | shape factor for conical contact profile (A_1)
    0.           : dt       | time increment

                                                                                                  imicro
      ************  Deformation-Stress Path Segments  **********   krotat           iflexc |             iplot
       (100000)   (10000)   (1000)    (100)      (10)      (1)       |            idump  | |ibodyf    ipts2 |
icontr| rate_11 | rate_22 | rate_33 | rate_12 | rate_13 | rate_23 igoal   finalv  ipts |  | | |  defdot  |   |
------|---------|---------|---------|---------|---------|---------|--|-|---------|----|-|--|-|--|------|----|--|
000000      0.        0.        0.         0.       0.        0.   70 0      10.     2 0  0 0  0    0.    0   0
100000      0.     -5.0e-7      0.         0.       0.        0.   70 0   10000.    50 0  0 0  0    0.    0   0
```

Figure 1: An example RunFile named LoadComp for a biaxial compression test with compression in the x_2 direction.

OVAL web site (see page ??) in the directory `oval/samples/results` with the file name `LoadComp`. I suggest that you use this file as a template for creating your own RunFiles.

The RunFile name will be used for assigning names to the various output files (Section 6 and Table 1). On Windows systems, you may want to give the RunFile name a `.txt` extension so that it will be more properly treated with word processors such as Word or Word Pad (see comments on page ?? and in item 9 on page 75).

A RunFile file is arranged in two parts:

1. a general information section consisting of the following 29 lines (see Section 8.1):
   - a single title line.
   - a series of 13 formatted lines that provide integer input.
   - a series of 15 formatted lines that provide floating point input.
2. five spacer lines of comments.
3. a deformation-stress path that consists of a series of formatted lines that describe each phase (segment) of the path (see Section 8.2). The program currently accepts up to 200 segments, although this limit can be changed with the source code parameter `lc1` in the source `common` file.

The contents of the first part, detailed in the next section, are contained in a series of 29 lines, each with a single input value at the beginning of the line. The third part, which specifies the deformation-stress path, is detailed in Section 8.2, page 32. Although the format specifier `f16.7` is used, Fortran allows the input data to be in either fixed (`F`) or exponential (`E` or `D`) formats with any number of significant digits, *provided that the data fits within the first 16 columns each line.*

8.1 RunFile: General information section

8.1.1 `title (a80)`

The `title` could include, perhaps, information on the nature of the simulation for your future reference. At present, the variable `title` is not used within the program, nor is it echoed to any of the output files.

8.1.2 `algori (i16)`

The program can be run with either of two DEM algorithms:

`algori=1` The conventional DEM algorithm (refer to 1979). The algorithm uses an implicit integration scheme. At present, this is the most robust of the two algorithms.

`algori=2` A new algorithm that the author has developed to self-monitor the progress of an intended pseudo-static simulation. With the standard algorithm (`algori=1`), the otherwise natural imbalance of forces on the particles can become excessive, particularly if the loading rate is too rapid. With the new algorithm (`algori=2`), several time steps are cycled within each deformation step. The cycling continues until the average force imbalance on a particle is within a threshold limit which constitutes a *near-equilibrium criteria*. At present the threshold is a particle force imbalance less than 1% of the average contact force magnitude (variable `chiavg.lt.chimax`). The number of cycles is currently programmed to be no less than 3 and no more than 101. See Sections 10.3.5, 10.3.9, and 10.3.15 for more information on the threshold limits that define the near-equilibrium criteria.

I recommend using `algori=1` with sparse assemblies (for example, if you are consolidating a gaseous assembly into a dense one) or when you are trying to capture the true dynamics of a deformation process (vibration studies, flow studies, etc.); but I recommend using either `algori=1` or `algori=2` for pseudo-static simulations with dense assemblies.

8.1.3 `ivers` (i16)

In early versions of the code, a RunFile consisted of 29 lines of general information. I later found the need for additional input information. Because Fortran is a rather inflexible language, the input value `ivers` was added, so the 29 lines could be extended, while preserving backward-compatibility with older RunFiles. For most simulations `ivers` can be set to 1.

`ivers=1` The original 29 lines of general information will be included in the RunFile.

`ivers=3` An additional 5 lines of integer information are included in the RunFile.

`ivers=4` An additional 5 lines of integer information are included in the RunFile, and an additional 8 lines of real information is included in the RunFile.

8.1.4 ncownt (i16)

When a flexible, tight-fitting boundary is used (Sections 7.2 and 8.2.8), it must be periodically updated, as the topology of the assembly is constantly changing. `ncownt` gives the frequency of updating the boundary. For example, if `ncownt=1`, the boundary is updated after every time step; if `ncownt=10`, the boundary is updated after every tenth step. When a flexible boundary is not being used, the input value of `ncownt` is ignored. If `ncownt=0` a flexible boundary is not being used, and the boundary will never be updated.

8.1.5 iout(2) (i16)

Currently not supported.

8.1.6 iout(3) (i16)

Currently not supported.

8.1.7 istart (i16)

The type of StartFile that will be used. The program supports three formats of StartFiles, which give the number of particles, particle type, and the initial particle arrangement (sizes, positions, etc.).

`istart=1` The initial particle arrangement will be given in a `D<RunFile>` file, henceforth referred to as a "D-file" (Section 9). This file is a text (ASCII) file (very portable).

`istart=2` The initial particle arrangement will be given in a `E<RunFile>` file, or "E-file". This file contains the same information as a D-file, but in a binary format (not portable, but smaller and faster). Also see Sections 8.1.8.

`istart=3` The entire initial state will be given in a `C<RunFile>` file, or "C-file". This binary "restart" file allows the current simulation to begin at the exact condition that was "dumped" at the end of a previous simulation. The restart file includes all positions, velocities, and contact information that allow the new run to begin where a previous run had left off. Note that with D- and E-files, the simulation will begin with the particles having zero velocities (or perhaps randomly assigned velocities, Section 8.1.25), and there will be no history of the contact forces. With restart C-files, the simulation will begin with velocities and forces carried over from a previous run. Also see Sections 8.1.8 and 8.2.7.

8.1.8 iend (i16)

The type of StartFile that will be created at the end of the simulation. The file that is created can later be used to define the initial condition for a future simulation (see Sections 6 and 8.1.7).

`iend=0` No file will be created at the end of the simulation.

`iend=1` An ASCII D<RunFile> file will be created, containing the final particle arrangement. See Section 9.

`iend=2` A binary E<RunFile> file will be created, containing the final particle arrangement. This file will contain the same information as a D-file but in a binary format.

`iend=3` A binary C<RunFile> file will be created, containing the entire end state of the simulation. This "dump" file can be used as a "restart" file to begin a future simulation at the exact ending state of the current simulation. Also see Sections 8.1.7 and 8.2.7.

8.1.9 idef (i16)

The value of `idef` determines the reference configuration of the assembly. `idef` is only of consequence when the StartFile is binary-type, restart C-file, and `idef` does not affect the results when the StartFile is a D-file or E-file.

`idef=0` When a C-file is being used to begin the simulation, then the reference configuration is carried over from the simulation from which the C-file was created. Deformations and deformation rates are relative to this older, carried-over configuration. That is, the strains that are reported as output are relative to an older configuration.

`idef=1` When a C-file is being used to begin the simulation, then the reference configuration is taken as the start of the current simulation. A zero-strain condition is reported at the start of the simulation. When a C-file is being used to start the simulation, the strains will be different at the start of the simulation than the strains that were reported at the end of the older simulation from which the C-file was created. Because of this difference, the results of a "restarted" simulation will differ from that of a simulation that continues to run past the restart point. The option `idef=1` is of no consequence when a D-file or E-file is being used to begin the simulation.

8.1.10 `iupdtm` (i16)

The frequency of updates to the linked list of near-neighbor particle pairs. You don't have to be too concerned about its value, as the program automatically determines if a more frequent update is required. A value of between 50 and 500 should be fine, but larger values will lead to somewhat faster computations. See Section 8.1.20.

8.1.11 `icirct` (i16)

Currently not supported.

8.1.12 `imodel` (i16)

The contact model.

`imodel=0` A linear contact model with friction (see Sections 8.1.15, 8.1.16, and 8.1.17).

`imodel=5` A Hertz-Mindlin contact model with friction. (see Sections 8.1.15, 8.1.16, and 8.1.17).

`imodel=6` A Jäger contact model with friction (2005; **?**). The Jäger contact is a generalized Cattaneo-Midlin-Deresiewicz contact for arbitrary sequences of loading and unloading in a three-dimensional setting. In this sense, it is far superior to the simple `imodel=5` model, which can only handle a single reversal in loading direction. Refer to Sections 8.1.15, 8.1.16, 8.1.17, and 8.1.26 for other input value that are required with the Jäger contact. Moreover, you will also need to adjust the value of the parameter `mlistJ` in the `common-X.X.XX` file and in the `subroutine Jager3D` of the `oval-X.X.XX.f` file. Parameter `mlistJ` has likely been set to a very low value in these locations, because a large value greatly in increases the size of the executable OVAL file. The value will need to be much larger when the Jäger model is used, and you should change `mlistJ` to, say, 500 or larger. The memory demand can be somewhat reduced with the input parameter `pdif`, Section 8.1.26.

8.1.13 `nplatn` (i16)

For boundaries of the external-particle type (Section 7.4), `nplatn` gives the number of files that must be read to provide information about the external particles, with one file per boundary. If you are not using this type of boundary, set `nplatn=0`.

8.1.14 `nloop1` (i16)

The minimum number of iteration time steps per deformation step. This value is only used when `algori=2`. If `nloop1` is zero and `algori=2`, a value of 3 is assigned to `nloop1`. If `algori=1`, then the input value of `nloop1` is ignored.

8.1.15 `kn` or `G` `f16.7`

With linear contacts (`imodel=1`), this input variable is the linear (spring) contact stiffness for determining the contact forces normal to contact surfaces. This stiffness is multiplied by the indentation at the particle contacts (i.e., half the overlap between two particles) to compute the normal contact force. As a result, the contact stiffness relative to the particle separation (overlap) is `kn`/2, so that the OVAL stiffness value will be half of that used in most DEM codes.

With Hertz-Mindlin contacts and Jäger contacts (`imodel=5` or `imodel=6`), this input variable is the shear modulus G of the particle material.

Although the format specifier `f16.7` is used, Fortran allows the input data to be in either fixed (`F`) or exponential (`E` or `D`) formats with any number of significant digits, provided that the data fits within the field width of 12 characters.

8.1.16 `kratio` (f16.7)

With linear contacts (`imodel=1`), this input variable is the ratio of two contact stiffnesses: the tangential stiffness divided by the normal stiffness.

With Hertz-Mindlin contacts and Jäger contacts (`imodel=5` or `imodel=6`), this input variable is the Poisson ratio of the particle material.

8.1.17 `frict` (f16.7)

The friction coefficient between particles.

`frict=0.` The contacts will be frictionless—friction will be "turned off." I sometimes use this mechanism to help densify a loose assembly.

`frict>0.` The contacts will be frictional with the given coefficient of friction.

8.1.18 `frictw` (f16.7)

The friction coefficient between particles and boundary walls (or boundary particles). See Section 7. This input value is ignored with periodic boundaries.

8.1.19 `rho` (f16.7)

The mass density of the particle material. See Section 8.1.29 for options to automatically assign a value of `rho`.

8.1.20 `search` (f16.7)

The threshold distance between two particles that will place them into a linked list of near-neighbors. To reduce the computation time, the subroutine that assembles the near-neighbor linked list is only occasionally called. The actual contact detection process, which is repeated with each time step, is only applied to this candidate linked list of near neighbors (Section 8.1.10). The threshold distance is equal to the dimensionless `search` value multiplied by the minimum particle radius. Larger values of `search` slow the contact detection process within every time step, since it will increase the length of the list of potential candidates, most of which will not actually be in contact. Larger values of `search`, however, will mean less frequent construction of the linked list of near-neighbors, a relatively slow process. Values of `search` between 0.20 and 0.80 seem to be appropriate.

8.1.21 `pcrit(1)` (f16.7)

A dimensionless coefficient of viscosity, which will be applied to the translational velocities of the particles. This coefficient represents a fraction of the critical damping $2\sqrt{mk}$, and the resulting viscous force is applied as a body force. When periodic boundaries are being used, this viscous damping is only applied to the particle velocities that are measured relative to the mean-field velocity. You will probably want to experiment with different values, with due attention to such performance parameters as `chi1`, `chi2`, `chi3`, `chi4`, and `psi` (pages 53–55).

8.1.22 `pcrit(2)` (f16.7)

A dimensionless coefficient of viscosity, applied to the rotational velocities of the particles (see the previous Section 8.1.21).

8.1.23 `pcrit(3)` (f16.7)

A dimensionless coefficient of viscosity, applied to the contact velocities of any pair of particles that are touching. This viscous force is applied as a contact force. The contact viscosity is "turned off" whenever frictional sliding occurs.

8.1.24 `xseed` (f16.7)

A seed for the random number generator. It is used for assigning initial random velocities to the particles. The seed is only used when `rmsvel>0`. See Section 8.1.25.

8.1.25 `rmsvel` (f16.7)

The average (root mean square) random particle velocity, assigned at the beginning of the simulation. Non-zero velocities can only be assigned when `algori=1` (Section 8.1.2). When a `rmsvel` is assigned, the particles are also given an initial angular velocity, on average about 10% of `rmsvel` divided by the mean particle radius (rather arbitrary, but this choice resides in "`subroutine init`" as `rotfac = 0.10d0`. Although velocities are randomly assigned, care is taken to assure that the initial momentum and angular momentum of the entire assembly is zero.

`rmsvel=0.` Do not assign initial random velocities to the particles. If `istart=1` or `istart=2`, the simulation will begin with the particles in an initially stationary state. When `istart=3`, the particle velocities will be carried over from a previous run regardless of the value of `rmsvel`.

`rmsvel>0.` A random velocity will be assigned to each particle, with the average (root mean square) random particle velocity equal to `rmsvel`. I sometimes use this feature to help densify an assembly by applying an artificial "vibration" technique. This option has no effect when the simulation is begun with a binary restart file (`istart=3`), since the velocities are carried over from a previous run. This option is only available when `algori=1` (Section 8.1.2).

8.1.26 `pdif` (f16.7)

When a Jäger contact model is being used (with `imodel=6`, see Section 8.1.12), this parameter can be used to reduce the memory demand of the model. It should be set to a value between 0 and twice the friction coefficient `frict`. When a Jäger contact model is not being used, with `imodel` not equal to 6, then this input value is ignored.

8.1.27 `tmax` (f16.7)

This input value sets a limit on the maximum time (or time steps) of the simulation. With some stress-control boundary conditions, OVAL will keep running until an input target stress has been reached (if ever). In some

situations, you may want set a limit on the number of time steps for the simulation. When `tmax` is 0 or negative, `tmax` will be ignored (i.e., with no control on the maximum length of the simulation).

8.1.28 `rfree3` (`f16.7`)

Currently not supported.

8.1.29 `dt` (`f16.7`)

The time step. The program will provide feedback on whether your time step is too large and will recommend a maximum time step, so you can just guess a trial time step and then adjust it later.

Several other options are available for establishing a time step, depending on the combined values of `dt` and `rho` (Section 8.1.19):

`dt>0, rho>0` If appropriate, your assigned values are used. OVAL will provide feedback on your assigned time step and the maximum recommended time step. (See the example output in Section 11.) If your time step exceeds this maximum, then OVAL will stop.

`dt=0, rho>0` The time step will be automatically set to a recommended time step. Your input density `rho` will be used.

`dt>0, rho=0` Your input time step will be used. An efficient density will be set to accommodate the input time step.

`dt=0, rho=0` The time step will be set to 1, and an efficient density will be set to accommodate this time step.

8.2 RunFile: Deformation-stress path section

The final section of a RunFile describes the manner in which either stresses or deformations are to be advanced. This section of the RunFile begins with five comment lines that are ignored by the program. These five lines are followed by a series of input lines, with each line specifying its *segment* of the desired deformation-stress path. The lines are arranged sequentially, with each line specifying a single segment of the deformation-stress path. Besides giving deformation-stress path information, these lines also determine the duration of each segment and specify supplementary actions to be taken at either the beginning or end of the segment. Each line contains 18 fields, arranged and formatted as follows:

```
format(i6,                  icontr
       6(1x,f9.6), 1x,      defrat
       i2,   1x,            igoal
       i1,   1x,            krotat
       f9.6, 1x,            finalv
       i4,   1x,            ipts
       i1,   1x,            idump
       i2,   1x,            iflexc
       i1,   1x,            imicro
       i2,   1x,            ibodyf
       f6.5, 1x,            defdot
       i4,   1x,            ipts2
       i2)                  jout
```

Note that the input fields are separated with the blank character (`1x`) and are arranged horizontally *on a single line*. Each line defines a single *segment* of the deformation-stress path. The content of each input field is detailed in the remainder of this section. I suggest that you use the sample RunFile "`LoadComp`" as a template for creating your own RunFiles.

8.2.1 `icontr (i6, 1x)`

The input variable `icontr` specifies the type of stress or deformation control. It does so by specifying six "components" of control.

The deformed shape of the assembly is described by the deformation gradient tensor $\mathbf{F}$, which we place alongside the corresponding components of the symmetric Cauchy stress tensor σ (I'm not suggesting that the two tensors are conjugate):

$$\mathbf{F} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ 0 & F_{22} & F_{23} \\ 0 & 0 & F_{33} \end{bmatrix} \quad \sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \quad \texttt{icontr} \rightarrow \begin{bmatrix} 1 & 4 & 5 \\ & 2 & 6 \\ & & 3 \end{bmatrix} \tag{1}$$

where compressive stresses are negative.

In the simplest form of `icontr`, you must control either the stress rate or the deformation rate for each of the six independent components (labeled 1 to 6 in the third matrix). For example, you could control the following combination of stress and deformation rates: $\dot{\sigma}_{11}$, $\dot{\sigma}_{22}$, $\dot{F}_{33}$, $\dot{F}_{12}$, $\dot{\sigma}_{13}$, and $\dot{F}_{23}$. In this simplest form, you must control either the stress rate or the deformation rate with each component; but you may *not* control *both* the stress and deformation rates of the same component, such as both $\dot{\sigma}_{13}$ and $\dot{F}_{13}$.

The input variable `icontr` specifies whether the deformation rate or the stress rate will be controlled for each of the six components. A `0` specifies

deformation control; a `1` specifies stress control. For example, an `icontr` value of `110010` would control the rates $\dot{\sigma}_{11}$, $\dot{\sigma}_{22}$, $\dot{F}_{33}$, $\dot{F}_{12}$, $\dot{\sigma}_{13}$, and $\dot{F}_{23}$, where the six digits of `icontr` are arranged in the order of components 11, 22, 33, 12, 13, and 23 (refer to the third matrix above for this mapping). The rates themselves are specified with `defrat` (Section 8.2.2).

Another example is shown in Fig. 1 on page 23. This RunFile contains two deformation-stress path segments. The first segment, with `icontr=000000`, is entirely deformation controlled. All deformation rates, `defrat`, are zeros, so that the segment is one of quiescent relaxation with no deformation. The second segment, with `icontr=100000`, maintains a constant stress σ_{22} (the rate $\dot{\sigma}_{22}$ is zero); it compresses the assembly at rate $\dot{F}_{22} = -5 \times 10^{-7}$; and it maintains zero deformations with the 33, 12, 13, and 23 components of $\mathbf{F}$.

OVAL permits some other forms of stress and deformation control.

- *Volume control*: when one or more of the first three digits of `icontr` is a "3", the corresponding deformations will be continually adjusted so that a given rate of volume change is maintained. The rate of volume change is given by the corresponding input variable `defrat` (Section 8.2.2). The volume rate is the computed as $d(F_{11}F_{22}F_{33})/dt$. The deformation rate is maintained by adjusting deformations of those components with the "3". For example if `icontr=133000`, then the 1st value of `defrat` gives the rate of stress change $\dot{\sigma}_{11}$; the 4th, 5th, and 6th values of `defrat` give the deformation rates $\dot{F}_{12}$, $\dot{\sigma}_{13}$, and $\dot{F}_{23}$; and the 2nd value of `defrat` gives the rate of volume change. In this situation, $\dot{F}_{22}$ and $\dot{F}_{33}$ will be continually adjusted to maintain that rate of volume change.

- *Mean stress control*: when one or more of the first three digits of `icontr` is a "2", the corresponding deformations will be continually adjusted so that a given rate of mean stress is maintained. For example if `icontr=022000`, then the 1st value of `defrat` gives the rate of deformation $\dot{F}_{11}$; the 4th, 5th, and 6th values of `defrat` give the deformation rates $\dot{F}_{12}$, $\dot{\sigma}_{13}$, and $\dot{F}_{23}$; and the 2nd value of `defrat` gives the rate of change of the mean stress. In this situation, $\dot{F}_{22}$ and $\dot{F}_{33}$ will be continually adjusted to maintain that rate of mean stress.

- *Deviator stress control*: when a single digit among the first three digits of `icontr` is a "4", the corresponding deformation will be continually adjusted so that a given rate of deviator stress is maintained. For example if `icontr=433000`, then the 1st value of `defrat` gives the rate of change of the deviator stress $q = d(\sigma_{11} - \max(\sigma_{22}, \sigma_{33}))/dt$, noting that the stress components are usually negative. In this example, the 4th, 5th, and 6th values of `defrat` give the deformation rates $\dot{F}_{12}$, $\dot{\sigma}_{13}$, and $\dot{F}_{23}$. The 2nd value of `defrat` gives the rate of volume change,

and $\dot{F}_{22}$ and $\dot{F}_{33}$ will be continually adjusted to maintain this rate of volume change.

8.2.2 `defrat (6(f9.6, 1x))`

These are the six rates of either deformation or stress, as specified by `icontr` (Section 8.2.1). A deformation rate corresponds to a component of the rate of change of the deformation gradient, $\dot{\mathbf{F}}$. The stress rates are the rates of change of components of the Cauchy stress tensor. Note that compressive stress is negative. At present, there is no provision for rotational springs at the particle contacts, so that the computed stress tensor components are very nearly symmetric ($\sigma_{ij} \approx \sigma_{ji}$, within the numerical precision of the computations), and, of course, there will be no couple stresses.

8.2.3 `igoal (i2, 1x)`

The input variables `igoal` and `finalv` determine the duration of the deformation-stress segment (see also Section 8.2.5). The duration is determined by monitoring just one of the six components of either stress or deformation (11, 22, 33, 12, 13, and 23) and whether that component has stepped across a given threshold value. The threshold value is specified with the input variable `finalv` (Section 8.2.5). The variable `igoal` is a 2-digit integer. The first digit is from `1` to `7`. Except when it is `7`, the first digit specifies which component will be monitored. The second digit is either `0`, `1`, or `4`. When the digit is `0`, the deformation threshold is monitored; when the digit is `1`, the stress threshold is monitored; and when the digit is `1`, a deviator stress threshold is monitored. The exception to this scheme is when the first digit is `7`, which specifies that the control segment will run for a fixed period of time. Several examples follow.

`igoal=51` The stress component (digit `1`) σ_{13} will be monitored (the fifth component, `13`, of stress). The particular deformation-stress control segment will finish when σ_{13} crosses the input threshold `finalv` from either above or below. For example, if the shear stress σ_{13} is 103.77 at the beginning of the control segment and `finalv` is 50.0, then the control segment will be finished when σ_{13} is reduced to 50.0 or less.

`igoal=20` The deformation component (digit `0`) F_{22} will be monitored (the second component, `22`, of deformation). If the deformation F_{22} is 0.745 at the beginning of the control segment and `finalv` is 0.800, then the control segment will be finished when F_{22} has increased to 0.800 or greater.

`igoal=70` The control segment will proceed for a given duration of time, as specified with `finalv`. The same result is achieved regardless of the second digit (`74`, `78`, etc.). I almost always use this scheme to specify the duration of a control segment. For example, if `igoal`=70, `finalv`=2.500, and `dt`=0.25, then the control segment will be exited after 10 time steps ($10 \times 0.250 = 2.500$).

`igoal=14` The deviator stress corresponding to stress 11 will be monitored: the stress $q = \sigma_{11} - \max(\sigma_{22}, \sigma_{33})$. The particular deformation-stress control segment will finish when this deviator stress crosses the input threshold `finalv` from either above or below.

When assigning values to `finalv`, the stresses σ_{11}, σ_{22}, and σ_{33} are usually negative (compressive).

It is can sometimes be difficult to foresee the direction in which a particular stress (or deformation) component will be moving, and so the specified component may actually move further away from the anticipated target `finalv`. For this reason, I usually just use a value of `70` for `igoal`.

8.2.4 `krotat (i1, 1x)`

You can select the dynamic behavior of the simulation by suppressing the particle movements or rotations.

`krotat=0` Neither rotations nor movements will be restricted (the usual situation).

`krotat=1` Particle rotations will be prevented, as in the simulation experiments of (1994). Only the mean-field rotations will be applied to the particles, as specified by the `defrat` rates $\dot{F}_{ij}$ (Sections 8.2.1 and 8.2.2).

`krotat=2` Particle translation will be prevented. Only the mean-field translations will be applied to the particles, as specified by the `defrat` rates $\dot{F}_{ij}$ (Sections 8.2.1 and 8.2.2).

`krotat=3` Both the particle rotations and the particle translations will be prevented. Only mean-field rotations and translations will be applied to the particles.

The actions of `krotat` apply only to the particular deformation-stress segment.

If `krotat` is not set to zero, you should avoid using `algori=2`, as this option can lead to very slow and inefficient simulations (Section 8.1.2). The

reason is that `algori=2` runs numerous relaxation cycles between each deformation increment, running relaxation cycles until the particles are in near-equilibrium. When movements are artificially restricted, the system will always be far from equilibrium.

8.2.5 `finalv` (`f9.6, 1x`)

See the discussion of `igoal`, Section 8.2.3.

8.2.6 `ipts` (`i4, 1x`)

The program OVAL can create *lots* of output. Simulations may require many thousands of time steps. The input variable `ipts` allows you to choose the frequency at which results are appended to the output files. For example, when `ipts` is `50` then results will be output to the A- and B-files every 50th time step (Sections 10.1–10.4).

8.2.7 `idump` (`i1, 1x`)

As has already been mentioned in regard to the input variable `iend`, the program allows you to "dump" a binary "restart" file at the *end* of a control segment (Sections 8.1.7 and 8.1.8). This binary file can later be used to start a new simulation from the exact conditions that were present at the time the restart file was created. You can use the field `idump` to dump a restart file in the middle of a simulation. Also see Sections 8.1.7 and 8.1.8.

`idump=0` Do not dump a binary restart file at the end of this control segment.

`idump=1` Dump a binary restart file at the end of this control segment.

The name of the output file will have following form:

```
C?-<RunFile>
```

where the "?" is a three digit number (e.g., `007`) that corresponds to the particular segment of the deformation-stress path at which the file was created.

8.2.8 `iflexc` (`i2, 1x`)

OVAL uses `iflexc` as a flag for creating tight-fitting boundaries and rigid-flat boundaries (Sections 7.2 and 7.3). The alternative external-particle boundaries are created with the use of `nplatn`. The input value `iflexc` is a two-digit integer: the first digit specifies the condition for the left and right boundaries; the second digit specifies the condition for the top and bottom

boundaries. An `iflexc=10` would create a stress boundary on top and bottom, but leave periodic boundaries on the left and right. An `iflexc=0` would leave all boundaries as periodic. The following boundary types, `iflexc=xx`, are available. These options are described in more detail in Sections 7.2 and 7.3.

`iflexc=0` Periodic boundaries (2D or 3D).

`iflexc=1` Stress control (2D only).

`iflexc=2` Displacement control tight-fitting boundary with free rotation of boundary particles (2D only).

`iflexc=3` Displacement control tight-fitting boundary with constrained rotation of boundary particles (2D only).

`iflexc=4` Displacement control tight-fitting boundary with free rotation of boundary particles and a frictional limit on the constrained translation of boundary particles (2D only).

`iflexc=5` Displacement control tight-fitting boundary with constrained rotation of boundary particles and a frictional limit on the constrained translation of boundary particles (2D only).

`iflexc=9` Rigid-flat boundaries for 2D or 3D assemblies (Section 7.3). For 3D assemblies, an `iflexc=99` will create rigid-flat boundaries on all six faces; an `iflexc=9` will create rigid-flat boundaries on the two x_1 faces; and an `iflexc=90` will create rigid-flat boundaries on the two x_2 faces.

Tight-fitting boundaries are created from an initial assembly having periodic boundaries. The initial assembly should be dense enough to have a complete load-bearing particle graph with well defined peripheral particles. To create tight-fitting boundaries, the first deformation-stress segment (line) in the RunFile should allow the initial assembly to equilibrate within its periodic boundaries (with, perhaps, a few hundred time steps). This first line would have `iflexc=0` The second deformation-stress segment (line) would specify the type of tight-fitting boundaries that should be created (`iflexc=xx`). The boundaries are created at the start of this deformation-stress segment. This second segment should have `icontr=000000` and a duration of several hundred time steps (`igoal=70` and `finalv>100`), so that the assembly can come to equilibrium within its new boundaries. Subsequent deformation-stress segments can be used to load the assembly.

8.2.9 `imicro (i1, 1x)`

OVAL can create a set of data files for use in post-processing the micro-level behavior of the assembly. These data files are described in Section 10.5. They can be used as input to your own Matlab, c, Fortran, Octave, Scilab, or R data analysis programs.

8.2.10 `ibodyf (i2, 1x)`

Currently not supported.

8.2.11 `defdot (f6.5, 1x)`

Currently not supported.

8.2.12 `ipts2 (i4, 1x)`

Currently not supported.

8.2.13 `iplot (i2, 1x)`

OVAL creates the binary G-files that become the input to the post-processor OVALPLOT for producing micro-level graphical depictions. The program OVALPLOT is described in the second part of this document. In particular, Section **??** describes the manner in which G-files should be created.

9 StartFile: The initial particle arrangement

The StartFile provides the initial particle arrangement, including the particle sizes, shapes, orientations, and positions. There are three types of StartFiles, with the type indicated by the leading character of the StartFile name: `C`, `D`, or `E`. Since only the `D`StartFile is readable as a text (ASCII) file, its contents will be described in this section. `E`StartFiles are just binary forms of `D`StartFiles. The nature and purpose of `C`StartFiles has already been described in Sections 6, 8.1.7, and 8.1.8.

Sample D-files can be found in the main `oval` directory in

`oval/samples/startfiles`

and descriptions of these assemblies are given in Section 12.

A D-file begins with either three or four lines that give general information about the assembly:

1. the type of particle (at present, the program does not allow the mixing of particle types within the same assembly). See Section 9.1.1.

2. the number of particles and the overall dimensions of the assembly (i.e. the distances between the periodic boundaries). See Section 9.1.2 and Fig. 5.

3. the shear offset distances (Section 9.1.3 and Fig. 5).

4. an angle β that describes the manner in which circular arcs are spliced together to create oval and ovoid particles (Fig. 6). This angle must be given in *degrees*. This line of input is *only required for the oval and ovoid particle types*. Section 9.1.4 discusses limitations on the input value of β.

These three (or four) lines are followed by information on each particle, with one line per particle. For example, the D-file for an assembly of 1002 circular particles might begin with

```
1
  1002  2.98994563276730430E+01  2.98446889993219070E+01  1.00000000000000000E+00
        0.00000000000000000E+00  0.00000000000000000E+00  0.00000000000000000E+00
  3.13454063710418570E-01  1.79352868741551070E+01  1.79266777574543750E+01
  4.43756654693070110E-01  1.27698119609280810E+01  2.52760548238972190E+01
etc.
```

The detailed contents of the first three or four lines are described in the following section. The remaining lines in the StartFile are described in Section 9.2.

9.1 Assembly data in D-StartFiles

9.1.1 `kshape` (i1)

The first column of the first input line should contain an integer of 1, 2, 3, 4, or 5, which will designate the type of particle.

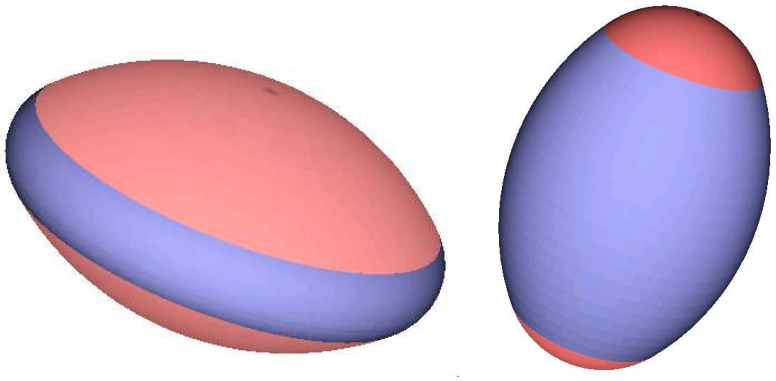
`kshape=1` circular (2D) disks

`kshape=2` oval (2D) disks. The ovals are composed of four circular arcs spliced together (Fig. 6). Although a more general variety of shapes can be formed from four (or more) arcs, the program currently supports only bi-symmetric ovals.

`kshape=3` elliptical (2D) disks. The code for this has not been recently tested and might not be stable.

`kshape=4` spheres (3D)

`kshape=5` ovoids (3D), a non-spherical shape that is composed of two spherical caps and a torus center (Fig. 2). The particle is smooth and strictly convex.



(a) Oblate (b) Prolate

Figure 2: Ovoid particles.

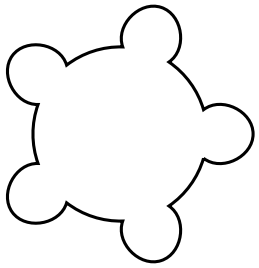


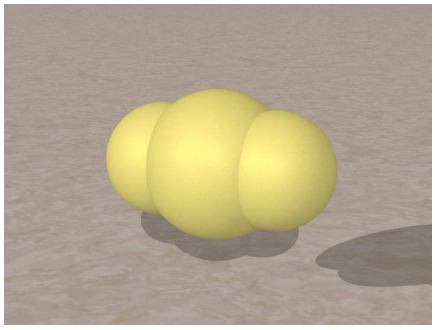
Figure 3: Nobby particle: `nobs=5`, `satrad=0.30`, and `cenrad=0.85`

`kshape=6` nobbies (2D), a non-circular shape that is composed of clusters of circles (Fig. 3). The particle is neither smooth nor convex.

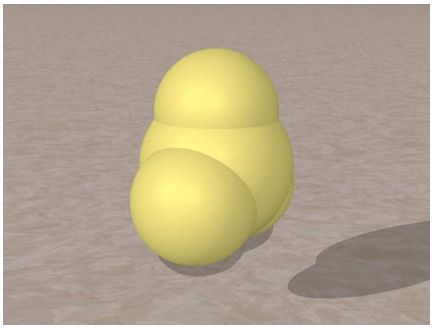
`kshape=7` bumpies (3D), a non-spherical shape that is composed of clusters of spheres (Fig. 4). The particle is neither smooth nor convex.

9.1.2 `np,xcell(1,1),xcell(2,2),xcell(3,3) (i6,3(1pd25.17))`

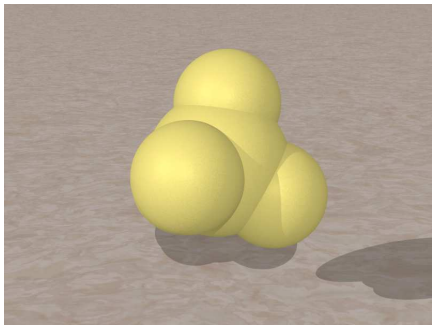
The number of particles, `np`, should appear within the first six columns, and the three dimensions of the assembly should be presented in double precision format, with 25 columns per dimension (Fig. 5). These dimensions are simply the spacings between opposing periodic boundaries. The value of `xcell(3,3)` must be given, even with 3D assemblies (any value will work, since it will be ignored in the simulation).



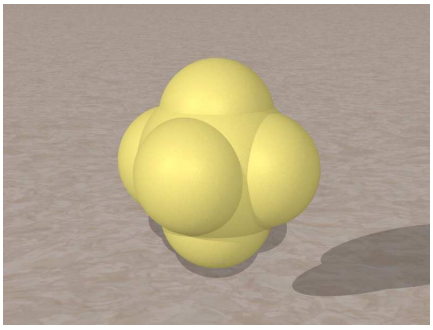
(a) `nbumps=2`, `satrad=0.7`, `cenrad=0.9`, `cirrad=0.75`



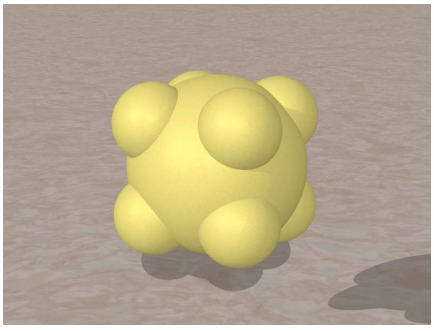
(b) `nbumps=3`, `satrad=0.8`, `cenrad=1.0`, `cirrad=0.7`



(c) `nbumps=4`, `satrad=0.6`, `cenrad=0.8`, `cirrad=0.7`



(d) `nbumps=6`, `satrad=0.7`, `cenrad=1.0`, `cirrad=0.6`



(e) `nbumps=8`, `satrad=0.4`, `cenrad=0.9`, `cirrad=0.8`

Figure 4: Bumpy particles.

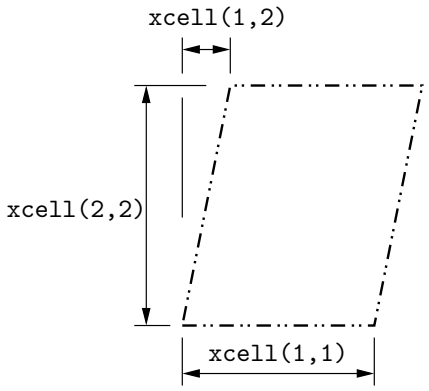


Figure 5: The `xcell(i,j)` dimensions for a 2D assembly

9.1.3 `xcell(1,2),xcell(1,3),xcell(2,3)` `(6x,3(1pd25.17))`

After six (blank) columns, the three shear offsets should be given in double precision format, with 25 columns per offset (see the offset `xcell(1,2)` in Fig. 5).

9.1.4 `beta` `(1pd25.17)` — oval and ovoid particles only!

When 2D ovals or 3D ovoids are being used, this fourth line contains the splice angle, `beta`, in *degrees* (Fig. 6). The angle β must be greater than zero and no greater than 90°. The particle aspect ratio will be limited by your choice of β (Sections 9.2.2 and 9.2.5). The height/width ratio must lie within the following bounds:

$$\frac{1-\cos\beta}{\sin\beta} < \alpha < \frac{\cos\beta}{1-\sin\beta} \; . \tag{2}$$

9.1.5 `nobs` or `nbumps` `(* integer)` — nobby (2D) and bumpy (3D) particles only!

When 2D nobby particles are being used, this line contains the integer number of satellite circles (arcs) that surround a central circle (see Fig. 3). The input variable `nobs` gives the integer number of satellite circles. These circles are equally spaced around the central circles. The satellite circles are centered on a "circumscribing circle" of radius 1.0.

When 3D bumpy particles are being used, this line contains the integer number of satellite spheres that surround a central sphere (see Fig. 4). Oval currently restricts `nbumps` to the following five values: 2, 3, 4, 6, or 8, with examples shown in Fig. 4. In each case, the satellite spheres of a bumpy particles will be centered on the surface of a "circumscribing sphere" (i.e.,

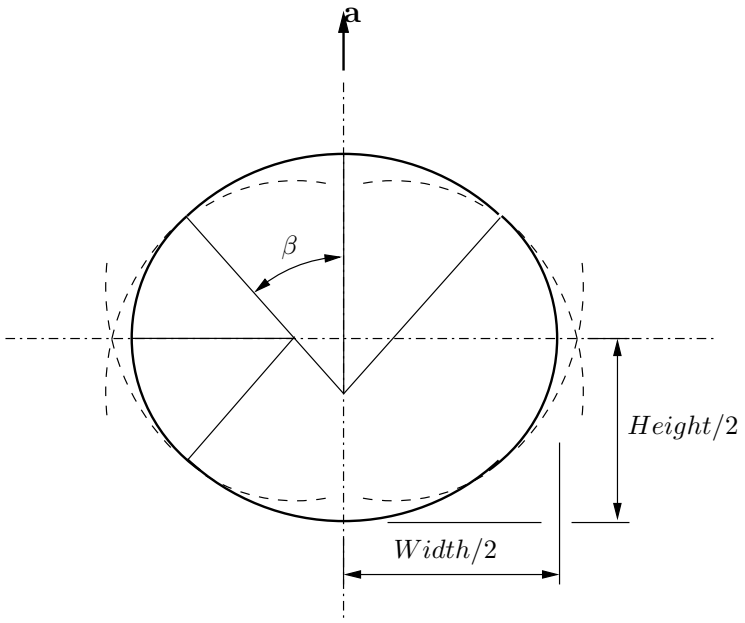


Figure 6: Geometry of an oval composed of four circular arcs.

the satellite spheres are not necessarily centered on the surface of the central sphere). The component spheres of bumpy (3D) particles have the following arrangements:

`nbumps=2` Two satellite spheres are place on opposite sides of the central sphere, diametrically opposed. The satellite spheres are centered on the surface of a "circumscribing sphere." See Fig. 4a.

`nbumps=3` Three satellite spheres are placed around the central sphere. The satellite spheres are centered on the vertices of an equilateral triangle that is circumscribed by the circumscribing sphere. That is, the satellite spheres are equally spaced along the equator of the circumscribing sphere. See Fig. 4b.

`nbumps=4` Four satellite spheres are placed around the central sphere. The satellite spheres are centered on the vertices of a tetrahedron that is circumscribed by the circumscribing sphere. That is, the satellite spheres are equally spaced around the full surface of the circumscribing sphere. See Fig. 4c.

`nbumps=6` Six satellite spheres are placed around the central sphere. The satellite spheres are centered on the vertices of a octahedron that is circumscribed by the circumscribing sphere. That is,

the satellite spheres are equally spaced around the full surface of the circumscribing sphere. See Fig. 4d.

`nbumps=8` Eight satellite spheres are placed around the central sphere. The satellite spheres are centered on the vertices of a cube that is circumscribed by the circumscribing sphere. That is, the satellite spheres are equally spaced around the full surface of the circumscribing sphere. See Fig. 4e.

Regardless of whether nobbies or bumpies are being used, this line of input must be followed by a line that gives relative sizes of the circles or spheres that comprise a particle, as in the following section (Section 9.1.6).

9.1.6 `satrad, cenrad, cirrad (* double precision)` — nobby (2D) bumpy and (3D) particles only!

These values are only used with nobby (2D) bumpy (3D) particles, and they are placed on a single line in double-precision format separated with spaces.

Every particle in an assembly will have the same shape, but they can have different sizes. Their common shape is specified with the values of `nobs`, `nbumps`, `satrad`, `cenrad`, and `cirrad`. These values give a "base" shape and size for the particles. The actual size of each particle is equal to the common "base size" multiplied by an individual scaling radius for each particle (Sections 9.2.6 and 9.2.7).

The input variables `satrad` and `cenrad` are the base sizes of the satellite and central circles or spheres of the nobby (2D) and bumpy (3D) cluster particles. The input variable `cirrad` is the radius of circumscribing sphere, as described in Section 9.1.5. OVAL currently does not support `cirrad` for 2D nobby particles: the satellite circles are centered on a circumscribing circle of radius 1.0.

9.2 Particle data in `D`-StartFiles

Following the general information at the head of a DStartFile, information is given on each particle, with one line per particle. The arrangement of this data depends upon the type of particle.

9.2.1 Circle particle data

Three fields give the radius, the x_1–location, and the x_2–location of the particle. The location coordinates refer to the center of the particle. Following the general information at the head of a DStartFile, this information is given for each circle, with each circle beginning on a new line. The data items are listed below. Each data field is in `1pd25.17` format, with spaces or a new

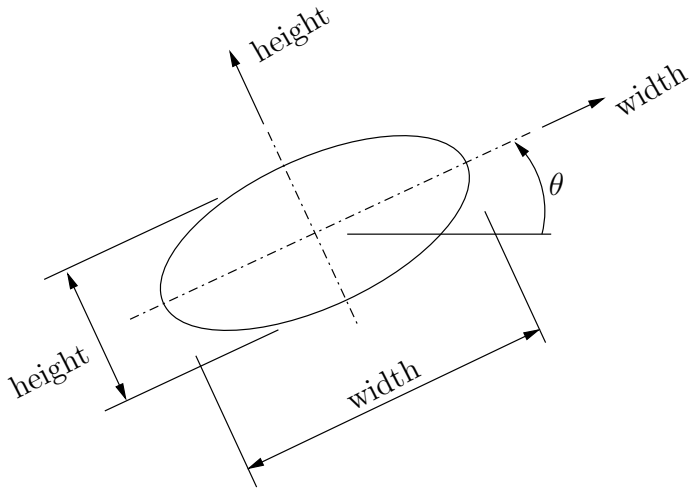


Figure 7: Orientation angle θ for elongated particles (2D ovals and ellipses).

line between the fields. In OVAL versions 0.6.0 and higher, the data can be provided in *free format*, so that data fields do *not* have to be aligned within certain columns. The example input on page 40 shows two lines of data for circular particles.

- Radius of the circle.
- x_1–location of the circle's center.
- x_2–location of the circle's center.

9.2.2 Oval particle data

Five fields give the oval width, the ratio of height divided by the width, the x_1–location, the x_2–location, and the orientation angle (in *degrees*) of particle, measured counterclockwise from the x_1-direction (Fig. 7). The location coordinates refer to the center of the particle. Following the general information at the head of a DStartFile, this information is given for each oval, with each oval beginning on a new line. The data items are listed below. Each data item is given in `1pd25.17` format, with spaces or a new line between the fields. In OVAL0.6.0, the data can be provided in *free format*, so that data fields do *not* have to be aligned within certain columns.

- Half-width of the oval. Note that the oval width may be greater or less than the height (with ellipses, the width must be greater than its height). The width is measured as shown in (Fig. 7), at an angle of θ counterclockwise from the horizontal x_1 axis. As input, you should give one half of the full particle width.

- Ratio of the height divided by the width of the particle (Fig. 7). For ovals, the ratio may be greater or less than one.
- x_1–location of the oval's center.
- x_2–location of the oval's center.
- Orientation angle θ of the oval's width (in degrees, Fig. 7).

9.2.3 Ellipse particle data

The code for this has not been recently tested and might not be stable. Five fields give the major radius, the ratio of the minor radius divided by the major radius (a number greater than zero, but no greater than one), the x_1-location, the x_2-location, and the orientation angle (in *degrees*) of the major axis, measured counterclockwise from the x_1-direction (Fig. 7). The location coordinates refer to the center of the ellipse. Following the general information at the head of a `DStartFile`, this information is given for each ellipse, with each ellipse beginning on a new line. The data items are listed below. Each data item is given in `1pd25.17` format, with spaces or a new line between the fields. In Oval0.6.0, the data can be provided in *free format*, so that data fields do *not* have to be aligned within certain columns. At present, the ellipse width must be greater than its height, with a height-to-width ratio less than one.

- Major radius of the ellipse.
- Ratio of the minor radius divided by the major radius of the particle (Fig. 7). For ellipses, the ratio must be greater than zero, but no greater than one.
- x_1–location of the ellipse's center.
- x_2–location of the ellipse's center.
- Orientation angle θ of the ellipse's width (in degrees, Fig. 7).

9.2.4 Sphere particle data

Four fields give the radius, the x_1–location, the x_2–location, and the x_3–location of the particle. The location coordinates refer to the center of the particle. Following the general information at the head of a `DStartFile`, this information is given for each sphere, with each sphere beginning on a new line. The data items are listed below. Each data item is given in `1pd25.17` format, with spaces or a new line between the fields. In Oval0.6.0, the data can be provided in *free format*, so that data fields do *not* have to be aligned within certain columns.

- Radius of the sphere.
- x_1–location of the sphere's center.
- x_2–location of the sphere's center.
- x_3–location of the sphere's center.

9.2.5 Ovoid particle data

Seven fields give the ovoid's revolved radius, the ratio of the axial radius divided by the revolved radius, the location coordinates, and the orientation angles γ_1 and γ_2 of the ovoid's axis of revolution, **a**. An ovoid is formed by rotating an oval (i.e. Fig. 6) about its central axis **a**. The location coordinates refer to the center of the particle. Following the general information at the head of a `DStartFile`, this information is given for each ovoid, with each ovoid beginning on a new line. The data items are listed below. Each data item is given in `1pd25.17` format, with spaces or a new line between the fields. In OVAL0.6.0, the data can be provided in *free format*, so that data fields do *not* have to be aligned within certain columns.

- Half of the transverse (revolved, half) width of the ovoid. This half-width is measured perpendicular to the central axis of revolution, **a**.
- Ratio of the axial height divided by the transverse width. The ratio can be greater or less than one, but it must be greater than zero. Ratios less than one are oblate; ratios greater than one are prolate (Fig. 2). Note that large aspect ratios (or small inverse ratios) require much more computation time. I would not recommend using a ratio greater than 2 or less than 0.5.
- x_1–location of the ovoid's center.
- x_2–location of the ovoid's center.
- x_3–location of the ovoid's center.
- γ_1 orientation angle (in radians) of the ovoid's axis of revolution, **a** (Fig. 8). This angle should be no less than zero and no greater than 90°.
- γ_2 orientation angle (in radians) of the ovoid's axis of revolution **a** (Fig. 8).

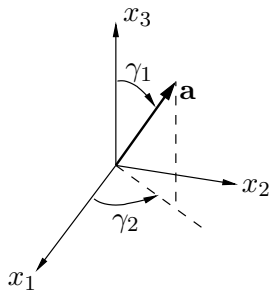


Figure 8: Orientation angles γ_1 and γ_2 for 3D ovoid particles. Vector **a** is the central axis of revolution of the ovoid (Figs. 6 and 7).

9.2.6 Nobby (2D) particle data

Four fields give the scaling radius, the x_1–location, the x_2–location, and the orientation angle (in degrees) of the particle. The location coordinates refer to the center of the particle. Following the general information at the head of a DStartFile, this information is given for each nobby, with each nobby beginning on a new line. The data items are listed below. Each data item is given in `1pd25.17` format, with spaces or a new line between the fields. In OVAL0.6.0, the data can be provided in *free format*, so that data fields do *not* have to be aligned within certain columns.

- Scaling radius of the nobby particle. This scaling radius is multiplied by the base radii, `satrad` and `cenrad`, as described in Section 9.1.6. These products give the actual sizes of the radii that comprise the nobby particle.
- x_1–location of the nobby particle's center.
- x_2–location of the nobby particle's center.
- θ_3 orientation angle (in degrees) of the particle. The center of one of the satellite particles is located at angle θ_3, measured counter-clockwise from the x_1–direction. The other satellite circles are equally spaced at angle $360°/$`nobs`.

9.2.7 Bumpy (3D) particle data

Eight fields give the scaling radius, the x_1–location, the x_2–location, the x_3–location, and a four-component unit quaternion of the particle's orientation. The location coordinates refer to the center of the particle. Following the general information at the head of a DStartFile, this information is given for each bumpy, with each bumpy beginning on a new line. The data items are

listed below. Each data item is given in `1pd25.17` format, with spaces or a new line between the fields. In OVAL0.6.0, the data can be provided in *free format*, so that data fields do *not* have to be aligned within certain columns.

- Scaling radius of the bumpy particle. This scaling radius is multiplied by the base radii (`satrad`, `cenrad`, and `cirrad`) as described in Section 9.1.6. These products give the actual sizes of the radii that comprise the nobby particle.
- x_1–location of the nobby particle's center.
- x_2–location of the nobby particle's center.
- x_3–location of the nobby particle's center.
- $\texttt{Qp}_1$ orientation quaternion of the particle.
- $\texttt{Qp}_2$ orientation quaternion of the particle.
- $\texttt{Qp}_3$ orientation quaternion of the particle.
- $\texttt{Qp}_4$ orientation quaternion of the particle.

10 Text output files from Oval

While OVAL is running, output is periodically written to two files: an A-file and a B-file. The frequency at which this occurs is specified with the input variable `ipts` (Section 8.2.6). These two files contain macro-data, such as stress and deformation. A-files and B-files are described in Sections 10.1–10.4. You can also produce F-files, which will contain micro-level data. Because F-files can be quite large, each creation of these files must be triggered by the input value `imicro` in the RunFile (Section 8.2.9). F-files are described in Sections 10.5–10.7.

10.1 A-files: macro-data for spreadsheets

A simulation will create a file named `A<RunFile>.txt`, which will be referred to as simply an "A-file". The fields in this text file are separated by Tab characters, so that A-files can be imported into a spreadsheet (note: *imported* not opened). When importing, you will want to specify the columns as being "tab separated." The spreadsheet is headed with some general information about the simulation, followed by a history of time, strains, and stresses. The strains are reported with the simple measure

$$F_{ij} - \delta_{ij} \tag{3}$$

where F_{ij} are components of the deformation gradient tensor. Note that the shear deformations are reported as shear angles, like F_{12}, or twice the shear strain (a γ-strain, not an ε-strain). Stresses and strains are reported at the interval `ipts`, which may differ for each segment of the deformation-stress path (Sections 8.2 and 8.2.6).

For 2D assemblies, the A-file includes information on the assembly's particle graph (1992; 1993; 1999). This information includes the numbers of vertices (particles that are included in the particle graph), edges (contacts between particles), and face (void cells).

For both 2D and 3D assemblies, the A-file includes information on the *fabric tensor* for the assembly (1982). We will refer to the fabric tensor with the symbol $\mathbf{A}$, which is defined with a sum over contacts within the assembly. Note that in an A-file, this tensor $\mathbf{A}$ is unfortunately labeled as "`F`", with the components `F(1,1)`, `F(2,2)`, etc.

$$A_{ij} = \frac{1}{N} \sum_{k=1}^{N} n_i^k n_j^k \ , \tag{4}$$

where $\mathbf{n}^k$ is the (unit vector) direction of the kth branch vector (contact), and N is the number of contacts in the assembly. When computing (4), N includes only those contacts between particles that each have at least 2 contacts (2D assemblies) or 3 contacts (3D assemblies).

For both 2D and 3D assemblies, the A-file also includes information on the fabric tensor $\mathbf{A}^s$ of only those ("strong") contacts that carry a greater-than-average force. Note that in an A-file, this tensor $\mathbf{A}$ is unfortunately labeled as "`Fs`", with the components `Fs(1,1)`, `Fs(2,2)`, etc. This tensor is defined as a sum over the strong contacts within the assembly:

$$A_{ij}^s = \frac{1}{N} \sum_{k \in \mathcal{S}} n_i^k n_j^k \ , \tag{5}$$

Where the set of contacts $\mathcal{S}$ is a set of contacts k:

$$\mathcal{S} = \{k : |\mathbf{f}^k| > \overline{f}\} \tag{6}$$

that have a force magnitude $|\mathbf{f}^k|$ greater than average:

$$\overline{f} = \sqrt{\frac{\sum_{k=1}^{N} |\mathbf{f}^k|^2}{N}} \tag{7}$$

The A-file also report the proportion v of strong contacts in the assembly.

An A-file contains several more columns of data. The labels of these columns explain their content, and many are given the same names as quantities that are described below for B-files (Section 10.2).

10.2 B-files: macro-data text files

The B<RunFile> (or just "B"-files) contain more information than A-files—not only stresses and deformations, but information that reflects the numerical performance of the simulation. For example, the B-file contains information that can characterize whether the simulation was nearly pseudo-static (`chi1`, `chi2`, `chi3`, `chi4`, and `knrgy`), the relative importance of viscous damping during the simulation (`chi3`, `chi4`, `viscbt`, and `viscct`), and whether the controlled stresses were maintained near their target values (`psi`). This information, although useful, is packed into a text (ASCII) file that can be somewhat difficult to read and understand. B-files are described in the following two sections.

10.3 B-files with 2D simulations

The first line in the B-file contains the following three pieces of information in `format(i4,2x,a50,2x,a20)`:

- an integer that indicates the format of the B-file,
- the name of the StartFile that was used for the simulation (Sections 6 and 9), and
- the version of the OVAL source code.

This line is followed by the results of each output cycle, which occur at the frequency `ipts` (see Section 8.2.6). For 2D simulations, the information for each output cycle is packed into just three lines, such as the following:

```
1.0320000E-04   5.948067E-05 -1.984000E-04  0.000000E+00  4.212188E-04     1623
  6.49E-04     -1.551593E+04 -2.071540E+04 -2.774535E+01  9.690910E+00 3.29E-07
  1.03E-03 0.00E+00 7.11E-04  6.519467E-03  5.346882E-02  2.655151E+00     3.00
```

The contents of these three lines correspond to the following variables (some of these are, in fact, arrays):

```
  timer         defout(1,1)   defout(2,2)   defout(1,2)    knrgy         ntacts
   chi1         stress(1,1)   stress(2,2)   stress(1,2)    pnrgy           psi
   chi2    chi3     chi4      viscbt        slidet         work1t        xloops
```

and in the following formats:

```
 1pe14.7,   1x,  1pe14.6,     1pe14.6,      1pe14.6,       1pe14.6,       i9,/,
 2x,1pe9.2, 4x,  1pe14.6,     1pe14.6,      1pe14.6,       1pe14.6,   1pe9.2,/,
 2x,1pe9.2, 1pe9.2, 1pe9.2,   1pe14.6,      1pe14.6,       1pe14.6,   0pf9.2
```

The various output values are now described.

10.3.1 `timer`

The accumulated time, which advances by an amount `dt` with each deformation step (Section 8.1.29).

10.3.2 `defout(i,j)`

The output values, `defout(i,j)`, are the difference between the deformation gradient matrix F_{ij} and the identity (Kronecker) matrix δ_{ij}, as in eqn 3 on page 50. For example, the output

```
defout(1,1) =  0.001
defout(2,2) = -0.002
defout(1,2) =  0.003
```

for a 2D assembly corresponds to the following deformation gradient:

$$\mathbf{F} = \begin{bmatrix} 1.001 & 0.003 \\ 0 & 0.998 \end{bmatrix} \tag{8}$$

The deformation gradient $\mathbf{F}$ is referenced to the initial assembly (except when the simulation is started with a C-StartFile, in which case, $\mathbf{F}$ is carried over from a previous run).

10.3.3 `knrgy`

The kinetic energy of particle motions per unit of the assembly's original volume. The kinetic energy is computed from both translational and rotational velocities of the particles:

$$\frac{1}{\text{Initial volume}} \times \frac{1}{2} \sum_{\text{Particles}} \left(m\bar{\mathbf{v}}^2 + I\bar{\boldsymbol{\omega}}^2 \right) , \tag{9}$$

where m and I are the mass and the mass moment of inertia of a particle, and $\bar{\mathbf{v}}$ and $\bar{\boldsymbol{\omega}}$ are the average velocity and angular velocity of a particle (the averages of the velocities at the two times $t - dt/2$ and $t + dt/2$). The energy `knrgy` is not really meaningful when `algori=2` (see Sections 8.1.2. Note that $F_{ij} = 0$ for $i > j$, as is the case in Fig. 5.

10.3.4 `ntacts`

The number of contacts. Here, a contact is shared by two particles. If you prefer to count a contact twice (once for each of the two particles, as in Table 4), then the number of contacts is, instead, `ntacts`×2.

10.3.5 `chi1`

One of four measures for determining whether the simulation is nearly pseudo-static. The property `chi1` is the average force imbalance on a particle divided by the average magnitude of a contact force. Small values signify that the particles were nearly in equilibrium during the deformation process. When `algori=2`, the variables `chi1` and `chi2` (Section 10.3.9) are used as a

near-equilibrium criteria for controlling the pace at which the deformation is allowed to proceed (Sections 8.1.2 and 10.3.15). If there are no contacts within the assembly, then `chi1` will be zero.

10.3.6 `stress(i,j)`

Components of the Cauchy stress tensor.

10.3.7 `pnrgy`

The elastic energy stored in the contact springs per unit of the assembly's original volume. For the simple linear contact mechanism, this energy is calculated as

$$\frac{1}{\text{Initial volume}} \times \frac{1}{2} \sum_{\text{Contacts}} \left[\frac{f_n^2}{k_n} + \frac{f_t^2}{k_t} \right] , \tag{10}$$

where f_n and f_t are the normal and tangential contact forces, and k_n and k_t are the normal and tangential contact stiffnesses at time t.

10.3.8 `psi`

This performance measure characterizes the fluctuation of the controlled stresses from their intended (target) values. When one or more digits in `icontr` are `1`'s, then a servo-control algorithm will attempt to control certain components of the Cauchy stress and maintain them at target values (see Section 8.2.1). The property `psi` is the sum of the deviations of the controlled stress components from their target values divided by the mean stress (either 1/2 or 1/3 σ_{kk}). If you are not controlling any of the stress components, then `psi` will be zero.

10.3.9 `chi2`

This is another measure of whether the simulation is nearly pseudo-static (see `chi1`, Section 10.3.5). The value `chi2` is the average *moment* imbalance on a particle divided by both the average magnitude of a contact force and the average particle radius. Small values signify that particles remained nearly in equilibrium during a simulation.

10.3.10 `chi3`

A measure of whether viscous contact damping could have a significant effect on the reported stresses and deformations. Its value is the viscous force at an average contact divided by the average contact force.

10.3.11 `chi4`

A measure of whether viscous body damping could be having a significant effect on the reported stresses and deformations. Its value is the viscous force on an average particle divided by the average contact force.

10.3.12 `viscbt`

The energy expended in viscous body damping per unit of the assembly's original volume (cumulative since the beginning of the simulation). This quantity may not be meaningful when `algori=2` (see Sections 8.1.2). The *increment* in `viscbt` for a time step dt is computed as follows:

$$\frac{1}{\text{Initial volume}} \times \sum_{\text{Particles}} \left[\eta^v \bar{\mathbf{v}} \cdot \bar{\mathbf{v}} dt + \eta^\omega \bar{\boldsymbol{\omega}} \cdot \bar{\boldsymbol{\omega}} dt\right] \ . \tag{11}$$

That is, the increment of work expended in body damping is equal the damping force ($\eta^v \mathbf{v}$) multiplied with the particle movement ($\mathbf{v}dt$). In this equation, η is the damping coefficient. The equation includes separate contributions of translational and rotational damping. The velocities $\bar{v}$ and $\bar{\boldsymbol{\omega}}$ are the averages for a particle at times $t - dt/2$ and $t + dt/2$.

10.3.13 `slidet`

The energy expended in frictional sliding per unit of the assembly's original volume (cumulative since the beginning of the simulation). The *increment* in `slidet` for a time step dt is computed as follows:

$$\frac{1}{\text{Initial volume}} \times \sum_{\substack{\text{Sliding} \\ \text{contacts}}} \left[\mu f^n \left|\bar{\mathbf{v}}^{\text{contact},\ t}\right| dt\right] \ , \tag{12}$$

where μ is the friction coefficient `frict`, f^n is the normal contact force, and $\bar{\mathbf{v}}^{\text{contact},\ t}$ is the tangential contact deformation. The velocity $\bar{\mathbf{v}}^{\text{contact},\ t}$ is an average of the velocities at times $t - dt/2$ and $t + dt/2$.

10.3.14 `work1t`

The work done by the "boundary stresses" per unit of the assembly's original volume. It is computed from the work rate

$$\frac{\text{Current volume}}{\text{Initial volume}} \int \sigma_{ij} \dot{F}_{ik} F_{kj} \, dt \tag{13}$$

and is cumulative from the beginning of the simulation. In this equation, σ_{ij} is the Cauchy stress.

10.3.15 `xloops`

When `algori=2`, several time steps will occur within each deformation step (Section 8.1.2). The program self-monitors the number of cycles that are required to achieve a near-equilibrium condition before advancing the assembly deformations. The property `xloops` is the average number of cycles that were required.

The program currently limits the number of cycles per deformation step to between `nloop1` and 101 (Section 8.1.14). When `xloops` is consistently reported as 3, the near-equilibrium criteria was met in three or fewer loops (see Section 8.1.2). When `xloops` is 101, then the near-equilibrium criteria were likely not met even after the final cycle. The threshold, near-equilibrium criteria is currently defined as a value of 0.5(`chi1` + `chi2`) be less than 1%.

10.3.16 `viscct`

This quantity appears in the A-files, but not in the B-files. The quantity `viscct` is the energy expended in viscous contact damping per unit of the assembly's original volume (cumulative since the beginning of the simulation). This quantity may not be meaningful when `algori=2` (Section 8.1.2). The *increment* in `viscct` for a time step dt is computed as follows:

$$\frac{1}{\text{Initial volume}} \times \sum_{\text{Contacts}} \left[\eta^n \bar{\mathbf{v}}^{\text{contact},\, n} \cdot \bar{\mathbf{v}}^{\text{contact},\, n} + \eta^t \bar{\mathbf{v}}^{\text{contact},\, t} \cdot \bar{\mathbf{v}}^{\text{contact},\, t} \right] dt \,, \tag{14}$$

where η^n and η^t are the coefficients of contact normal and contact tangential damping, and $\bar{\mathbf{v}}^{\text{contact},\, n}$ and $\bar{\mathbf{v}}^{\text{contact},\, t}$ are the components of contact deformation velocities in the normal and tangential directions. That is, the contribution to `viscct` of a single contact is the contact damping force $\eta \bar{\mathbf{v}}^{\text{contact}}$ multiplied by the displacement increment $\bar{\mathbf{v}}^{\text{contact}} dt$.

10.4 B-files with 3D simulations

As with 2D simulations, the first line of the B-file contains a file-type identifier, the name of the StartFile, and the version of the Oval source code (see Section 10.3). This line is followed with results for each output cycle. With 3D simulations, the information for each output cycle is packed into five lines, such as the following:

```
2.6000000E-05   5.313745E-06 -5.000000E-05  0.000000E+00  6.161242E-05   5168
      9.65E-05  0.000000E+00  0.000000E+00  0.000000E+00  4.398784E+02
      1.13E-04 -5.684897E+05 -6.075969E+05 -5.778599E+05  2.126770E-02
      0.00E+00 -2.138418E+04  5.103270E+03 -1.097130E+04  2.574228E+01
      6.82E-05       5.10E-07  2.169865E-02  0.000000E+00         30.00
```

These contents correspond to the following variables (some of these are, in fact, arrays):

	File name	Content
2D	Fa?<RunFile>	Assembly size
	Fb?<RunFile>	Particle data
	Fc?<RunFile>	Contact data
	Fd?<RunFile>	Void cell data
3D	Fa?<RunFile>	Assembly size
	Fb?<RunFile>	Particle data
	Fc?<RunFile>	Contact data

Table 2: Contents of the various F-files. Note that the "?" character in a file name (Table 2) is a 3-digit number (e.g., 005) that corresponds to the particular segment of the deformation-stress path in which the F-file was created (Section 8.2.9).

```
 timer          defout(1,1)   defout(2,2)   defout(3,3)    knrgy       ntacts
     chi1       defout(1,2)   defout(1,3)   defout(2,3)    pnrgy
     chi2       stress(1,1)   stress(2,2)   stress(3,3)    slidet
     chi3       stress(1,2)   stress(1,3)   stress(2,3)    work1t
     chi4            psi         viscbt        viscct       xloops
```

and in the following formats:

```
 1pe14.7,  1x,  1pe14.6,      1pe14.6,      1pe14.6,       1pe14.6,       i7,/,
 1pe15.2,       1pe14.6,      1pe14.6,      1pe14.6,       1pe14.6,          /,
 1pe15.2,       1pe14.6,      1pe14.6,      1pe14.6,       1pe14.6,          /,
 1pe15.2,       1pe14.6,      1pe14.6,      1pe14.6,       1pe14.6,          /,
 1pe15.2,       1pe14.2,      1pe14.6,      1pe14.6,       0pf14.2
```

These output fields were described in the previous section (10.3).

10.5 F-files: micro-data text files

F-files contain information on the positions of all particles and the status of all contact forces. These files are created during an OVAL simulation by setting `imicro=1` within a deformation-stress segment of the RunFile (Section 8.2.9). With `imicro=1`, a set of F-files will be created at the *start* of the particular deformation-stress segment. As many as four separate F-files will be produced, with each containing a different type of information (Table 2). The files can be quite large: for an assembly of 1000 particles, a set of 2D F-files is about 300kBytes. Note that the "?" character in a file name (Table 2) is a 3-digit number (e.g., 005) that corresponds to the particular segment of the deformation-stress path in which the F-file was created (Section 8.2.9).

The contents of these text files are described in the following two sections. You will probably want to use a data analysis package to open, read, and analyze their data (e.g. Matlab, Octave, Scilab, R, etc.). The various F-files only contain information on the *status* of the assembly at a single instance;

they do not provide velocities or other rates. If such rates are of interest, then you should use a RunFile that will produce two sets of F-files, with the two files separated by just a few time steps.

10.6 F-files for 2D assemblies

Four F-files are created at once (Table 2), and they are described in the following four subsections. F-files for 3D assemblies are described in Section 10.7. All F-files are created in `subroutine micro`.

10.6.1 Fa-files for 2D assemblies

These small files give the size of the assembly, the average deformation gradient relative to the start of the simulation, and other general information. The file consists sixteen lines in the format `i3,/,1pe14.7,/, 9(3(1pe25.17),/),i2,/, 3(1pe17.9,/)`:

```
   file_identifier
   timer
   xcell(1,1)  xcell(1,2)  xcell(1,3)
   xcell(2,1)  xcell(2,2)  xcell(2,3)
   xcell(3,1)  xcell(3,2)  xcell(3,3)
     def(1,1)    def(1,2)    def(1,3)
     def(2,1)    def(2,2)    def(2,3)
     def(3,1)    def(3,2)    def(3,3)
  stress(1,1) stress(1,2) stress(1,3)
  stress(2,1) stress(2,2) stress(2,3)
  stress(3,1) stress(3,2) stress(3,3)
  kshape
  kn
  kratio
  frict
  beta
```

The `file-identifier` simply identifies the version of the F-files, in the event that the file content or format is modified at a later date (This identifier was introduced in OVAL0.6.5). The cell dimension `xcell(i,j)` were illustrated in Fig. 5, page 43. The `def(i,j)` deformations are components of the deformation gradient $\mathbf{F}$. For 2D assemblies, only four of the nine components `xcell`, `def`, and `stress` are meaningful.

The meanings of `kshape`, `kn`, `kratio`, and `frict` are described in Sections 9.1.1, 8.1.15, 8.1.16, and 8.1.17. With Hertz-type contacts (`imodel=5` or `imodel=6`, see Section 8.1.12), the shear modulus G is written in place of `kn`, and the Poisson ratio ν is written in place of `kratio`.

With nobby (2D) particles (`ishape=6`) three additional lines are written: `nobs`, `satrad`, and `cenrad`, as in Sections 9.1.5 and 9.1.6. With nobby (3D) particles (`ishape=7`) four additional lines are written: `nbumps`, `satrad`, `cenrad`, and `cirrad`, as in Sections 9.1.5 and 9.1.6.

10.6.2 Fb-files for 2D assemblies

These files contain information on the size and position of each particle. Each line gives data for a single particle, with the lines arranged by particle number (e.g. line number 3 is for particle 3). The format of each line is `i7,4(1pe17.9),1(1pe18.9)`, with the fields as follows:

Field 1	`hv`, a pointer to the DCEL (see Section 10.6.3)
Field 2	Particle half width (Fig. 6, page 44)
Field 3	Aspect ratio, Height/Width (Fig. 6, page 44)
Field 4	x_1 position
Field 5	x_2 position
Field 6	θ orientation in radians (Fig. 7, page 46)

Note that when `hv=0`, the particle is in contact with no other particles. The positions x_1 and x_2 correspond to the particle centers. See Section 9.2.

10.6.3 Fc-files for 2D assemblies of convex particles

These fields contain information on every contact within the assembly. The content of this file will differ for convex particles (circles, ellipses, and ovals) and non-convex particles (nobbies). This section applies to convex particles; non-convex particles are described in the next section.

The format of each line is `4(i7),2(i8),2(1pe17.9),5(1pe13.5)` (with Hertz-Mindlin contacts, the value of `Tstar` in format `1pe13.5` is included at the end of a line). The information in this file will allow you to reconstruct the entire topology of the 2D assembly (some data from the Fb- and Fc-files will also be needed) and to navigate within this topology. Doing this efficiently requires a rather ingenious data structure called a Doubly-Connected-Edge-List (DCEL). Although I plan to include a better description in a future edition of this document, for now you should refer to the text by (1985), which describes the DCEL and how to use it to navigate the topology of a 2D planar graph.

The first six fields in each line contain the following information on a contact: `V1`, `V2`, `F1`, `F2`, `P1`, and `P2`. As an example, Table 3 shows five rows and the first six columns in an Fc-file (Fig. 9).

- Row 3 in an F-file corresponds to contact 3 (see row 3 in Table 3).

V1	V2	F1	F2	P1	P2
2233	1	1	4	6172	2
1	3985	1	2	3	6393
1	225	2	3	4	849
1	345	3	4	1	1260
690	2	5	8	2381	7

Table 3: An example DCEL table (Fig. 9).

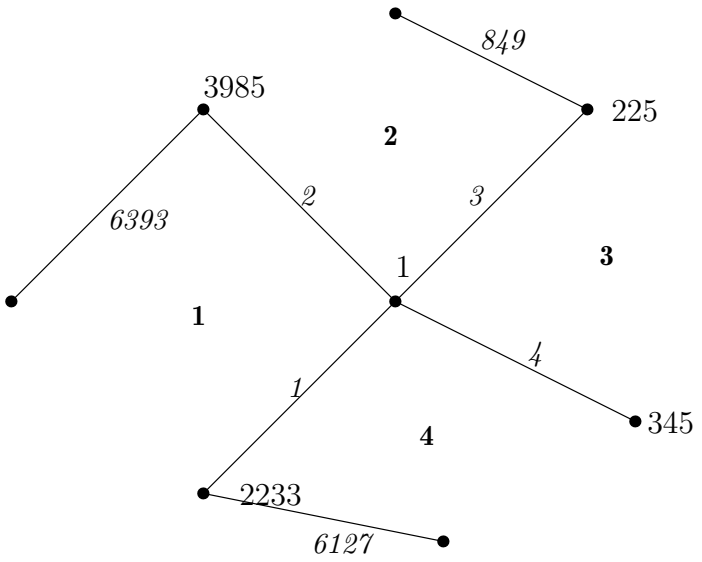


Figure 9: The particle graph associated with the DCEL of Table 3. The dots represent particle centers. Lines represent the contacts between particles.

- Contact 3 is between particle number 1 (`V1`) and particle number 225 (`V2`) (i.e., rows 1 and 225 in the corresponding Fb-file).
- Void cell 2 (`F1`) lies on one side of this contact, and void cell 3 lies on its other side (`F2`). See Section **??** for a further description of void cells.
- Pointer `P1` (= 4) points to a contact (row 4), which is also connected to `V1` (particle number 1) and lies directly clockwise around `V1` relative to the row 3 contact.
- Pointer `P2` (= 849) points to the contact (row 849) that is also connected to `V2` (= particle 225) and lies clockwise around `V2` relative to the third (row 3) contact.

The `hv` data in an Fb-files point to the starting (header) contact for a particle (Section 10.6.2). With this data structure, you will be able to identify all of the contacts that are connected to an arbitrary particle. You will also be able to identify all contacts and particles that lie around the perimeter of a polygonal void cell. That is, you can construct both the particle graph and its dual (1992). The `hf` data in an Fd-file points to the starting (header) contact for a void cell (Section 10.6.5).

The final seven columns in an Fc-file give the following data:

- `branch(1)` and `branch(2)`: the horizontal and vertical component of the branch vector that connects the centers of particles `V1` and `V2` (directed toward `V2`). Note that the format of these two fields is `1pe17.9`.
- `c_eta(1)` and `c_eta(2)`: the horizontal and vertical components of the unit normal vector, directed outward from the surface of particle `V1` at the contact location.
- `fnold1(1)`: the magnitude of the contact force component that acts normal to the contact surface—a positive value for compressive contact force.
- `ftold(1)` and `ftold(2)`: the horizontal and vertical components of the contact force tangential to the contact surface. The force acts upon particle `V1`.
- `Tstar`: For Hertz-Mindlin contacts only (`imodel=5`, Section 8.1.12). See (1988).

10.6.4 Fc-files for 2D assemblies of non-convex particles

With non-convex particles, such as nobbies, two particles can have multiple contacts between each other. The Fc-file gives information identifies the two particles and their component parts (satellite and central circles) that are touching. Each line in the Fc-file has format `4(i7),2(i8),2(1pe17.9), i3,4(i7,i3,i3)` and contains the following information:

- The first six fields contain `V1`, `V2`, `F1`, `F2`, `P1`, and `P2`. These are described in Section 10.6.3.
- The next two fields contact `branch(1)` and `branch(2)` as described in Section 10.6.3.
- `ihit`: the number of contacts between these two particles.
- Four triples of integers. With non-convex particles, such as nobbies, two particles can have multiple contacts between each other. Each triple corresponds to one of these contacts. Each triple is composed of a pointer to a position within the list of an "Ff" file and two identifiers of the component circles that that are touching (one identifier for the first particle `V1`, the other identifier for the second particle `V2`). The identifier is an integer ranging between 0 and `nobs`: the zero corresponding to the central circle, numbers 1 and above corresponding to the satellite circles (arcs). If a pair of particles has fewer than four contacts, then zeros will appear in the unused triples.

10.6.5 Fd-files for 2D assemblies

The lines of this file contain a single integer in `i7` format. The `hv` integer on each line is a pointer to the DCEL for a single void cell (Section 10.6.3). The lines are given in order (for example, line 14 gives the `hv` value for void cell number 14).

10.6.6 Ff-files for 2D assemblies

These files are only created for non-convex particles, such as nobbies. Each line in this file gives information on a single contact. Note that a pair of particles can share multiple contacts. As described in Section 10.6.4, Each pair of contacting particles will be represented with a single line in an Fc-file. For non-convex particles, the Fc-file will identify the two particles (the locations of these particles are given in the Fb-file), the branch vector between the particles' centers, number of contacts between the two particles, and triple-integers that identify each of these contacts. The first integer in

a triple points to a line in the Ff-file. The contents of the Ff-file depends on the contacts' displacement-force model (`imodel`, Section 8.1.12).

For nobby particles with a linear-frictional contact model (`imodel=0`), this line has format `4(1pe17.9)` and contains the following information about the contact:

- The two components of the contact's normal vector (directed outward from the first particle, `V1`).
- The normal force (compression is positive).
- The three components of the contact's tangential force (acting upon the first particle, `V1`).

For nobby particles with a simple Hertz-Mindlin model (`imodel=5`), this line has format `5(1pe17.9)` and contains the following information about the contact:

- The two components of the contact's normal vector (directed outward from the first particle, `V1`).
- The normal force (compression is positive).
- The three components of the contact's tangential force (acting upon the first particle, `V1`).
- `Tstar`.

For nobby particles with the Jäger contact model (`imodel=6`), this line has format `7(1pe17.9)` and contains the following information about the contact:

- The two components of the contact's normal vector (directed outward from the first particle, `V1`).
- The normal force (compression is positive).
- The three components of the contact's tangential force (acting upon the first particle, `V1`).

10.7 F-files for 3D assemblies

Four F-files are created at together (Table 2), and they are described in the following three subsections: Fa-files (Section 10.7.1), Fb-files (Section 10.7.2), Fc-files (Section 10.7.3), and Ff-files (Section 10.7.4).

10.7.1 Fa-files for 3D assemblies

Same data as with 2D assemblies (Section 10.6.1).

10.7.2 Fb-files for 3D assemblies

These files contain information on the size and position of each particle. Each line gives data for a single particle, with the lines arranged by particle number (line number 3 in the file is for particle 3).

For 3D assemblies of spheres, the format of each line is `4(1pe17.9),3(1pe18.9e3)`, with the following fields for each sphere:

Field 1	radius
Field 2	x_1 position of the particle center
Field 3	x_2 position of the particle center
Field 4	x_3 position of the particle center
Field 5	θ_1 angular orientation of the particle (radians)
Field 6	θ_2 angular orientation of the particle (radians)
Field 7	θ_3 angular orientation of the particle (radians)

For 3D assemblies of ovoids, the format of each line is `7(1pe17.9),3(1pe18.9e3)`, with the following fields for each ovoid:

Field 1	transverse (revolved) half width (Section 9.2.5)
Field 2	aspect ratio: axial height / transverse width
Field 3	x_1 position of the particle center
Field 4	x_2 position of the particle center
Field 5	x_3 position of the particle center
Field 6	γ_1 orientation angle (in degrees) of the particle's axis, (Fig. 8, Section 9.2.5)
Field 7	γ_2 orientation angle (in degrees) of the particle's axis, (Fig. 8, Section 9.2.5)
Field 8	θ_1 angular orientation of the particle (radians)
Field 9	θ_2 angular orientation of the particle (radians)
Field 10	θ_3 angular orientation of the particle (radians)

For 3D assemblies of bumpies, the format of each line is `4(1pe17.9),4(1pe18.9e3)`, with the following fields for each ovoid:

Field 1	scaling radius (Sections 9.1.6 and 9.2.7)
Field 2	x_1 position of the particle center
Field 3	x_2 position of the particle center
Field 4	x_3 position of the particle center
Field 5	1st component of the unit orientation quaternion
Field 6	2nd component of the unit orientation quaternion
Field 7	3rd component of the unit orientation quaternion
Field 8	4th component of the unit orientation quaternion

10.7.3 Fc-files for 3D assemblies

Each line in this file gives information on a single contact.

For 3D assemblies of spheres, the format of each line is `2i7,,3(1pe17.9)`, `4(1pe13.5)`, with the following fields for each sphere (with Hertz-Mindlin contacts, the value of `Tstar` in format `1pe13.5` is included at the end of a line):

- `V1` and `V2`: the two particle numbers
- `branch(1)`, `branch(2)`, and `branch(3)`: the components of the branch vector that connects the centers of particle `V1` and particle `V2` (directed toward `V2`). Note that the format for these three fields is `1pe16.8`.
- `fnold1(1)`: the magnitude of the contact force component that acts normal to the contact surface—a positive value for compressive forces.
- `ftold(1)`, `ftold(2)`, and `ftold(3)`: the three components of the portion of the contact force that is tangential to the contact surface. The force acts upon particle `V1`.
- `Tstar`: For Hertz-Mindlin contacts only (`imodel=5`, Section 8.1.12). See (1988).

For 3D assemblies of ovoids, the format of each line is `2i7,3(1pe17.9)`, `10(1pe13.5)`,with the following fields for each ovoid (with Hertz-Mindlin contacts, the value of `Tstar` in format `1pe13.5` is included at the end of a line):

- `V1` and `V2`: the two particle numbers.
- `branch(1)`, `branch(2)`, and `branch(3)`: the components of the branch vector that connects the centers of particle `V1` and particle `V2` (directed toward `V2`). Note that the format for these three fields is `1pe17.9`.
- `rx_i(1)`, `rx_i(2)`, and `rx_i(3)`: The components of a vector from the center of particle `V1` to the center of the contact point between the two particles.

- `fnold1(1)`: the magnitude of the contact force component that acts normal to the contact surface—a positive value for compressive contact force.

- `ftold(1)`, `ftold(2)`, and `ftold(3)`: the three components of the portion of the contact force that is tangential to the contact surface. The tangential force acts upon particle `V1`.

- `c_eta(1)`, `c_eta(2)`, and `c_eta(3)`: The three components of the unit vector that is the outward normal of particle `V1` at the contact point.

- `Tstar`: For Hertz-Mindlin contacts only (`imodel=5`, Section 8.1.12). See (1988).

For 3D assemblies of bumpies, the format of each line is `2i7,3(1pe17.9)`, `i3,6(i7,i3,i3)`, with the following fields for each bumpy:

- `V1` and `V2`: the two particle numbers.

- `branch(1)`, `branch(2)`, and `branch(3)`: the components of the branch vector that connects the centers of particle `V1` and particle `V2` (directed toward `V2`). Note that the format for these three fields is `1pe17.9`.

- `ihit`: the number of contacts between these two particles.

- Six triples of integers. With non-convex particles, such as bumpies, two particles can have multiple contacts between each other. Each triple corresponds to one of these contacts. Each triple is composed of a pointer to a position within the list of an "Ff" file and two identifiers of the component spheres that that are touching (one identifier for the first particle `V1`, the other identifier for the second particle `V2`). The identifier is an integer ranging between 0 and `nbumps`: the zero corresponding to the central sphere, numbers 1 and above corresponding to the satellite spheres.

10.7.4 Ff-files for 3D assemblies

These files are only created for non-convex particles, such as bumpies. Each line in this file gives information on a single contact. Note that a pair of particles can share multiple contacts. As described in Section 10.7.3, Each pair of contacting particles will be represented with a single line in an Fc-file. For non-convex particles, the Fc-file will identify the two particles (the locations of these particles are given in the Fb-file), the branch vector between the particles' centers, number of contacts between the two particles, and triple-integers that identify each of these contacts. The first integer in

a triple points to a line in the Ff-file. The contents of the Ff-file depends on the contacts' displacement-force model (`imodel`, Section 8.1.12).

For bumpy particles with a linear-frictional contact model (`imodel=0`), this line has format `4(1pe17.9)` and contains the following information about the contact:

- The normal force (compression is positive).
- The three components of the contact's tangential force (acting upon the first particle, `V1`).

For bumpy particles with a simple Hertz-Mindlin model (`imodel=5`), this line has format `5(1pe17.9)` and contains the following information about the contact:

- The normal force (compression is positive).
- The three components of the contact's tangential force (acting upon the first particle, `V1`).
- `Tstar`.

For bumpy particles with the Jäger contact model (`imodel=6`), this line has format `7(1pe17.9)` and contains the following information about the contact:

- The three components of the contact's normal vector (directed outward from the first particle, `V1`).
- The normal force (compression is positive).
- The three components of the contact's tangential force (acting upon the first particle, `V1`).

11 Screen output from Oval

As a simulation is running, information is printed to the screen, which can help to monitor the performance of the run. This information can, of course, alternatively be redirected from the screen to a file. The following is an example of the introductory information that might appear on the screen at the beginning of a simulation:

```
Program OVAL:  version            oval-0.5.41.f
c Matthew R. Kuhn 2001, Licensed under the GPL, version 2

The program was compiled under the following parameters:
   1) 2D or 3D problems. (mdim1=3)
   2) Circular, spherical, elliptical, oval, and ovoid particles. (mpiece=4*mp)
```

```
   3) A maximum of 10020 particles. (mp)
Errors will occur if your input data is otherwise (but you can always make
changes to the common-0.5.41 file and recompile).

  Name of the RunFile:
LoadComp                 <-- your input here
  Name of the StartFile:
Dsphere_1800             <-- your input here

 **** Warning ****.
 * The input value of rho was 0.
 * A mass will be automatically assigned.

 **** Warning ****.
 * The input value of dt was 0.
 * A time step will be automatically assigned.

  Your time step is:                1.00000E+00
  The maximum advised time step is: 1.25000E+00

  The assigned particle mass is:    9.76563E+00

  Spherical, 3D particles

  Number of particles     =    1800
  Initial void ratio      =  0.535828
  Initial solids fraction =  0.651115
  Initial porosity        =  0.348885
  Volume of the cell      =  2.248245E+03

  Initial number of contacts       =    5130
  Average ratio of overlap/diameter =  3.186E-04
```

In this example, the names of the two input files are `Load1` and `Dcircls_1002`. The introductory information is followed by a table of diagnostic information that is periodically updated at the interval `ipts`, as specified in the RunFile (Section 8.2.6). This table will look something like the following:

```
Some diagnostic information during this run:
iout    timer istep nupd  ipt2 xloops   chi1      chi2       psi    sweep
   0  0.0000E+00  1    1  18108   1.0  0.00E+00  0.00E+00  0.00E+00  0.00
   1  2.0000E+00  1    1  18108  21.5  1.10E-02  7.73E-03  0.00E+00  0.00
   2  4.0000E+00  1    1  18108   3.0  9.81E-03  7.04E-03  0.00E+00  0.00
   3  6.0000E+00  1    1  18108   3.0  8.79E-03  6.34E-03  0.00E+00  0.00
   4  8.0000E+00  1    1  18108   3.0  7.98E-03  5.74E-03  0.00E+00  0.00
   5  1.0000E+01  2    1  18108   3.0  7.14E-03  5.15E-03  0.00E+00  0.00
   6  5.8000E+01  2    2  18109   2.9  2.42E-03  1.77E-03  3.68E-06  0.00
   7  1.0800E+02  2    2  18109   3.0  3.55E-04  3.77E-04  1.39E-06  0.00
   8  1.5800E+02  2    3  18109   3.0  2.68E-04  3.38E-04  1.17E-06  0.00
```

Most items in the table were described in Section 10.3. The integer `istep` is the current deformation-stress segment (from the RunFile, Section 8.2). The integer `nupdat` is the number of near-neighbor searches that have been

File Name	Particle Type	No. of Particles	Void Ratio	Coord. Number[5]	Dimensionless Overlap
`Dcircls_1002_2`[1]	circles	1002	0.18042	3.820	3.10×10^{-4}
`Dcircls_4008_2`[1]	circles	4008	0.17911	3.813	3.19×10^{-4}
`Dovals_1002_1d`	ovals	1002	0.18382	3.784	2.32×10^{-4}
`Dovals_1002_2d`	ovals	1002	0.12890	4.880	1.58×10^{-4}
`Dovals_4008_1d`	ovals	4008	0.18479	3.777	2.10×10^{-4}
`Dovals_4008_2d`	ovals	4008	0.12788	4.733	1.72×10^{-4}
`Dsphere_1800`	spheres	1800	0.53583	5.700	3.19×10^{-4}
`DOblate_1800`[2]	ovoids	1800	0.41520	8.214	1.65×10^{-4}
`DProlate_1800`[3]	ovoids	1800	0.41223	8.438	2.01×10^{-4}
`DObProlate_1800`[4]	ovoids	1800	0.41367	8.351	2.01×10^{-4}

[1]The assemblies `Dcircls_1002` and `Dcircls_4008` in OVAL version `0.4.0` have been replaced. These older assemblies were slightly anisotropic.
[2]Oblate ovoids with randomly assigned aspect ratios. The aspect ratio is uniformly distributed between 0.65 and 1.00.
[3]Prolate ovoids with randomly assigned aspect ratios. The aspect ratio is uniformly distributed between 1.00 and 1.60.
[4]A combination of oblate and prolate ovoids with randomly assigned aspect ratios. The aspect ratio is uniformly distributed between 0.65 and 1.60.
[5]The coordination number is computed as twice the number of contacts divided by the number of particles.

Table 4: Attributes of several sample assemblies

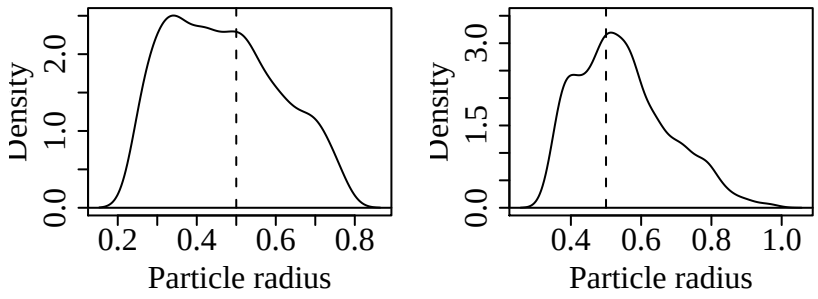
performed (see Section 8.1.20). The integer `ipt2` is the current length of the near-neighbor linked list (Section 8.1.20). The value `xloops` is the average number of iteration loops (time steps) per deformation step. When `algori=1`, then `xloops` will always be one. The values , , and indicate the numeric performance of the run (see Sections 10.3.5, 10.3.9, and 10.3.8 and the other sections referenced from there). The value `sweep` is the average number of iterations per torus-torus contact when ovoids are being used.

12 Sample assemblies for Oval

You can download sample StartFile assemblies, which are given in a D-file (text) format, from the web site given on page **??**. The sample StartFiles should be located in your `oval` directory in

`samples/startfiles`

The primary attributes of these assemblies are shown in Table 4. Each assembly is roughly square (or cubical) and with an isotropic fabric. They were created by isotropically compressing a sparse assembly with friction



(a) 2D assemblies (b) 3D assemblies

Figure 10: Histograms of particle radii for 2D and 3D assemblies

turned off.

The assemblies contain a range of particles sizes. The dimensions of the particles and the entire assemblies have been scaled so that the mean particle size D_{50} in each assembly is 1.00. (Here, we speak of the mean particle size D_{50} in the usual sense of geotechnical engineering: a "median" diameter that partitions the assembly into two sets of particles, so that each set has an equal cumulative mass.) A density plot (normalized histogram) of particle radii for both 2D and 3D assemblies is shown in Fig. 10. The histogram is centered on the median size $0.50D_{50}$. In Fig. 10a, the radii of non-circular 2D particles refers to their mean radii, $(\text{Height} + \text{Width})/4$ (Fig. 7, Section 9.2). In Fig. 10b, the radii of non-spherical 3D particles refers to their mean radii, $(\text{Height} + 2 \cdot \text{Width})/6$ The distribution of aspect ratios for oval and ovoid assemblies are described in the footnotes of Table 4.

The void ratio is a measure of the packing density of a granular assembly (Table 4). The assemblies of circles, spheres, and ovoids are fairly dense. Both loose and dense assemblies of ovals are provided. The coordination numbers shown in Table 4 are computed as twice the number of contacts divided by the number of particles.

Among the attributes listed in Table 4, the dimensionless overlap probably has the greatest effect on the speed and performance of a simulation. The dimensionless overlaps, which are quite small, were computed by dividing the average overlap at the particle contacts by the mean particle size, D_{50}. Small overlaps more closely resemble those in real granular materials, which are often composed of hard granules. During simulations, however, small overlaps require slower deformation rates to assure the near quasi-static progression of particle rearrangements.

In addition to the files in Table 4, the website includes a directory

Start File Name	Particle Type	No. of Particles	Run Time
`Dcircls_4008`	circles	4008	19m 52s
`Dovals_1002_2`	ovals	1002	9m 51s
`Dovals_4008_2`	ovals	4008	43m 10s
`Dsphere_1800`	spheres	1800	19m 21s

Table 5: Execution times for simulations with the RunFile shown in Fig. 1, page 23 and various StartFile assemblies (Table 4). The times are with an Intel Pentium III 450MHz processor.

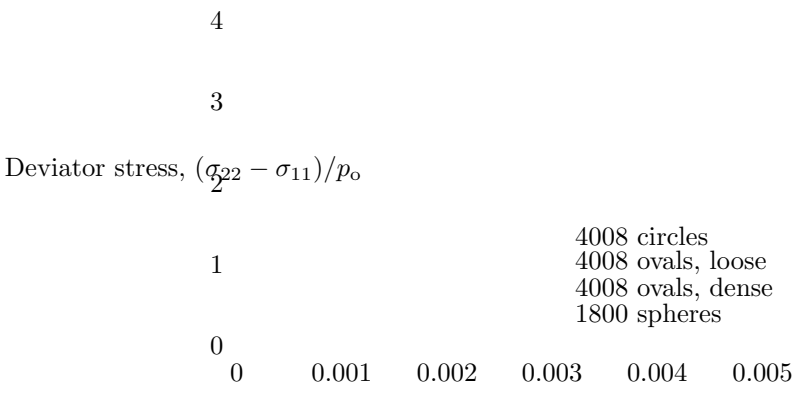
`samples/startfiles/Series_circls_1002/` that contains 100 assemblies of 1002 circular particles. The particle size distribution in each assembly is the same as that of `Dcircls_1002_2` in Table 4 and Fig. 10a. Each assembly has exactly the same particle sizes and the same void ratio (=0.174257), but the assemblies have different particle arrangements. As a result of this difference, the files will have modest differences in the number of contact, dimensionless overlap, and initial stress. The assemblies were constructed with the same process, but the the particle radii were shuffled among the particles before the diffuse assembly was compacted.

13 Example simulations using Oval

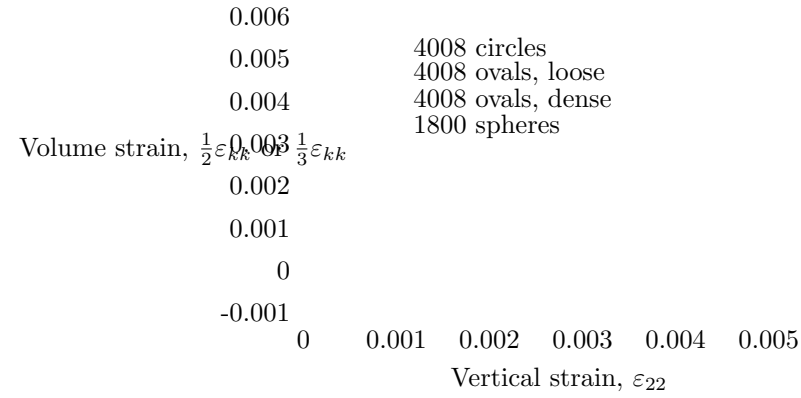
As an example simulation, consider the RunFile named `LoadComp` shown on page 23. After a brief initial period in which the assembly is allowed to equilibrate, the assembly is vertically compressed while maintaining constant horizontal stress ($\dot{F}_{22} < 0$, $\dot{F}_{11} = 0$). For 3D assemblies, the deformations were under plane strain conditions ($\dot{F}_{33} = 0$). The results for all of the larger assemblies are shown in Fig. 11 and are archived at the web site (page **??**). These results can also be found in the your `oval` directory in

`samples/results`

The deviator stress in Fig. 11 has been normalized by dividing by the initial mean stress, p_o. Note the quite large difference in strengths of the loose and dense assemblies of ovals. The loose assembly (created quite by accident) is much weaker than the assembly of circular discs, even though the two assemblies have nearly the same void ratio. The strength of the 3D assembly of spheres is much larger than that of the 2D assembly of circles, which is due, in part, to the plane strain conditions during compression of the 3D assembly. The execution times for the three tests are shown in Table 5, using an Intel Pentium III 450MHz machine and executable binaries produced by the pgf77 Linux compiler. As can be seen in the table, the execution time is roughly proportional to the number of particles. Oval particles require



(a)



(b)

Figure 11: Results for both 2D and 3D materials: deviator stress and volumetric behavior.

about twice the time of circular particles. Spherical particles also require about twice the time of circular particles.

14 Some advice on using Oval

When Oval is properly used, the program is efficient and provides repeatable results. The program can be maddening, however, when it is unknowingly being stretched beyond its limits. You may want to consider the following advice.

1. Slow is (usually) better. When choosing deformation or stress rates with the input values `defrat(i,j)`, the program's performance can be greatly improved by choosing appropriately slow rates (Section 8.2.1 and 8.2.2). What is an appropriate rate? If the rate is too slow, you will needlessly waste time (days, weeks, months) waiting for your simulation to finish. If the rate is too fast, the program will either fail to maintain quasi-static conditions (when `algori=1`, Section 8.1.2), will run excessively slow while attempting to establish quasi-static conditions (when `algori=2`), or will crash. A common error message upon crashing is the following:

   ```
   An illegitimate contact in subroutin lister.
   ```

   This error occurs when the particle velocities are excessive, causing two particles that were previously not even in the linked-list of near-neighbors to come into contact within a single time step (Section 8.1.20). Oval will not stop running, but if this message repeatedly occurs or the results become erratic, you will probably want to reduce the deformation rate. (I am fairly tolerant of this error when I am not particularly interested in the accuracy of the results, for example when I am preparing an assembly from a sparse arrangement of particles.)

   The proper deformation rate depends primarily on (and is almost proportional to) the average overlap among particles in their initial configuration. If the overlaps are too large (relative to the particle radii), the simulation will not be very realistic. With smaller average overlaps, slower deformation rates will be required to maintain nearly quasi-static conditions. Note that the relative overlaps are fairly small for the sample assemblies that are included in the Oval package (Table 4, page 69).

   The best way to determine a suitable deformation rate may be by trial and error. If you are testing an initially dense assembly, choose `algori=2` and try a few `defrat` values. The first deformation-stress

control segment, however, should not involve any deformation. A beginning period of quiescence is required to allow the initial particle arrangement to come to near-equilibrium. The first segment should, therefore, have `icontr=000000` and should be long enough (`igoal=70`, with sufficient time `finalv`) to allow a enough steps for the initial particle arrangement to equilibrate (Sections 8.2.3 and 8.2.1).

The second and subsequent stress-deformation control segments are where you can test the deformation rate. As the program runs and output appears on the screen, wait until these segments are entered (see the `istep` values on the screen output, page 68), and then monitor the values of `chi1`, `chi2`, and `xloops` (Sections 10.3.5, 10.3.9, and 10.3.15). The option `algori=2` provides a minimum of 3 equilibrating time steps per deformation step, with a maximum of 101 time steps (Section 8.1.2). If `xloops` is consistently 3.0, you can probably increase the trial deformation rate. If `xloops` is consistently much greater than 3.0, then the deformation rates are probably too high. I usually try to keep `xloops` near 3 or 4 and `chi1` and `chi2` below 0.005.

2. For diffuse, sparse assemblies, use `algori=1` (Section 8.1.2). This will be the case if you are trying to compact a gaseous assembly into a dense one. For dense assemblies, use `algori=2`, as this will enforce a self-regulating mechanism for maintain nearly quasi-static conditions.

3. I sometimes give the particles initial velocities (`rmsvel>0`) to help in densifying an initially loose particle arrangement. As has already been stated, slow is usually better. If you get the message,

   ```
   An illegitimate contact in subroutin lister.
   ```

   then you have probably assigned an `rmsvel` value that is too large.

4. When starting a simulation with a D-file or E-file, the particles will likely not be in an equilibrium configuration, since these files only specify the particle positions and provide no information about the contact forces (see `istart=3`, Section 8.1.7). This means that the initial calculation of contact forces will give zero tangential force, a condition not likely to produce equilibrium. The first deformation-stress segment, therefore, should not involve any deformation. Instead, a beginning period of quiescence is required to allow for the initial particle arrangement to come to near-equilibrium. The first segment should, therefore, have `icontr=000000` and should be long enough (`igoal=70`, with sufficient time `finalv`) to allow a few time steps for the initial particle arrangement to equilibrate (Sections 8.2.3 and 8.2.1). The second and subsequent stress-deformation control segments are where you can start the desired deformation process.

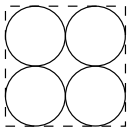
5. Binary E-file and C-file formats might not be portable between different platforms or operating systems.

6. If you are assigning initial random velocities to the particles (an input value `rmsvel`≠`0`), do not assign velocities that are too large. You will probably need to reduce the value of `rmsvel` if you get the error message:

   ```
   An illegitimate contact in subroutine lister.
   ```

   See item 1 above.

7. OVAL does not work when the assembly contains too few particles, say fewer than nine 2D particles or twenty-seven 3D particles. This problem is related to the use of periodic boundaries. As an example, consider a square arrangement of four particles of equal size with this arrangement:

   Each particle touches a neighboring particle twice: once within the core assembly and once again across a periodic boundary. OVAL's underlying data structure allows for a single contact per particle pair. This problem is avoided with a larger number of particles. (Thanks to Csilla and Tamás.)

8. Do not try to control a boundary stress when the boundary stress is zero. For example, an input value `icontr=11100` will not work with a diffuse, gaseous assembly (Section 8.2.1).

9. If you get an error message that your input files can not be properly read, you will want to carefully check the formatting of the file's input fields. Microsoft Windows users may have problems with hidden characters that can be embedded in files when using Word and Word Pad. You will probably want to install a genuine text processor and avoid using word processors.

10. If at the start of a simulation you get the unexpected message, "`Name of a platen file:`", then you have probably given `nplatn` a nonzero value in your RunFile (see Section 8.1.13).

15 Change Log

This section documents the changes that have been made between various version of OVAL and OVALPLOT.

15.1 Oval–0.4.0 to Oval–0.6.0

Added features:

- Added the 3D non-spherical ovoid particle (Sections 9.1.1, 9.2.5, 10.7.2).
- Added the "krotat" option of either allowing or preventing particle rotations and/or particle translations (Section 8.2.4).
- Converted to list-directed input for D-files. This allows for much less restrictive input files (for example, input files created with spreadsheets).
- Added more information printed to the screen at the start of a run.
- Added the option of automatically computing the mass and/or the time step to optimize performance (Sections 8.1.29).
- Added the option of `nloop1` into the input RunFile (Section 8.1.14).

Enhancements:

- Improved the near-neighbor search algorithm for ellipse, oval, and ovoid particles. These changes should reduce the search time by up to a factor of 10.
- Added several hundreds of comments to the source code.

Bug fixes:

- With 3D assemblies, print header line in B-files.
- Corrected output value of xloops in 3D B-files.
- Moved the code for repositioning ovals within the subroutines "integ1" and "integ2". This change will be necessary for handling non-periodic boundaries
- Previously, the particles were placed back into the main periodic cell in subroutin "lister" whenever they drifted outside the main cell. What seemed like a nice act of basic housekeeping actually produces problems with OvalPlot. I eliminated this feature.
- Corrected an error in the handling of periodic boundaries in the near-neighbor search subroutine "lister".

15.2 Oval–0.6.0 to Oval–0.6.1

Added features:

- Added information contained in the A-file: fabric tensor $\mathbf{A}$; fabric tensor of strong contacts $\mathbf{A}^s$; proportion of strong contacts; numbers of edges, faces, and vertices in the particle graph, etc. (Section 10.1).

Enhancements:

- Changed the code for single-precision (4-byte) to double-precision (8-byte) for all floating point numbers.
- Changed the labeling scheme for `C?`, `Fa?` files (Sections 8.2.7, 10.5, and Table 2)
- Changed the names of F-files from `F1<...>` to `Fa<...>`, etc. (Sections 10.6 and 10.7)

Bug fixes:

- Fixed bug that prevented the creation of F-files for non-spherical particles.

15.3 Oval–0.6.1 to Oval–0.6.2

Added features:

- F-files now give output of the ortientation angles of the particles.

Enhancements:

- Increase the precision of output of some fields in F-files.
- Increased the precision of contact inquiry for ovoids.

Bug fixes:

- Fixed calculation of the average overlap among particles.

15.4 Oval–0.6.2 to Oval–0.6.3

Bug fixes:

- Changes to contact inquiry algorithm for ovoids.

15.5 Oval–0.6.3 to Oval–0.6.4

Enhancements:

- Add more information about the simulation parameters in the "`Fa`"-files (the `beta` angle, `fn`, `ft`, `frict`, and stress).
- Changed all angles in the "`Fb`"-files to radians (γ_1, γ_2, and θ).
- Changes to screen output during an OVAL run to accommodate larger simulations.

Bug fixes:

- Slight change in algorithm for contact damping.

15.6 Oval–0.6.4 to Oval–0.6.5

Enhancements:

- Increased precision (digits) in the "`Fa`"-files.
- Improved stability of the ovoid contact detection algorithm.

15.7 Oval–0.6.5 to Oval–0.6.8

Enhancements:

- Report the effective void ratio in the A-files.
- Added most of the B-file information to the A-files.
- Added the possibility of U- and V-files.

Bug fixes:

- Corrected error in defining `kshape` when reading a dump file.
- Limit the number of error message that can be printed to the `errfile`. Large numbers of error could previously fill a file system.
- Corrected an error in overflows of `chi1` and `chi2`.

15.8 Oval–0.6.8 to Oval–0.6.10

Enhancements:

- Added `ivers` to extend RunFiles.
- Added Hertz-Mindlin contact.
- Added flexible boundary and several forms of rigid boundaries.
- Added the option `iexact` for exactly integrating the kinetics of ovoid particles
- Added external code for generating random numbers.
- Added information to A-files: the number of sliding contacts, etc.

16 References

Bardet, J. P. (1994). "Observations on the effects of particle rotations on the failure of idealized granular materials." *Mech. of Mater.*, 18(2), 159–182.

Cundall, P. A. and Strack, O. D. L. (1979). "A discrete numerical model for granular assemblies." *Géotechnique*, 29(1), 47–65.

Jäger, J. (2005). *New Solutions in Contact Mechanics*. WIT Press, Southampton, UK.

Kuhn, M. R. (1999). "Structured deformation in granular materials." *Mech. of Mater.*, 31(6), 407–429.

Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction*. Springer-Verlag, New York.

Satake, M. (1982). "Fabric tensor in granular materials." *Proc. IUTAM Symp. on Deformation and Failure of Granular Materials*, P. A. Vermeer and H. J. Luger, eds., A.A. Balkema, Rotterdam, 63–68.

Satake, M. (1992). "A discrete-mechanical approach to granular materials." *Int. J. Eng. Sci.*, 30(10), 1525–1533.

Satake, M. (1993). "New formulation of graph-theoretical approach in the mechanics of granular materials." *Mech. Mater.*, 16, 65–72.

Thornton, C. and Randall, C. W. (1988). "Applications of theoretical contact mechanics to solid particle system simulation." *Micromechanics of Granular Materials*, M. Satake and J. Jenkins, eds., Elsevier Science Pub. B.V., Amsterdam, The Netherlands, 133–142.