Oval and OvalPlot: Programs for analyzing dense particle assemblies with the Discrete Element Method

Matthew R. Kuhn*, kuhn@up.edu

June 1, 2006
Versions Oval-0.6.18 and OvalPlot-0.4.2

1 Introduction

Oval is a program for DEM analysis of particle assemblies. The program is freely distributed under the open source GNU General Public License, Version 2. The Discrete Element Method (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 (Cundall and Strack 1979). Perhaps most important, the method allows extracting the 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. The program Oval 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*. Oval can also be used to compact a diffuse assembly into a denser state.

The program handles both two- and three-dimensional simulations with particles that are either circles (2D), ellipses (2D), ovals (2D), spheres (3D), or a special non-spherical "ovoid" particle (3D). The program has been compiled and run on both Unix and Windows systems.

OvalPlot is a program for graphically displaying the results of 2D Oval simulations. The program displays a spatial representation of the micro-mechanical processes that occur as the assembly is deformed. OvalPlot is intended for two-dimensional assemblies only.

If you plan to use Oval and OvalPlot, please send me an email message so that you are kept informed of future upgrades to the package. If you

*Dept. of Civil Engineering, University of Portland, 5000 N. Willamette Blvd., Portland, OR 97203 USA, Fax 503–943–7316, Tel 503–943–7361, kuhn@up.edu.

find errors, please let me know. If you modify the source code, please give the code a new name. For example, if you make changes to the original file `oval-0.7.114.f`, then rename the file `oval-0.7.114-`*your_name*`.f`.

The programs OVAL and OVALPLOTare free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. These programs are distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place—Suite 330, Boston, MA 02111-1307, USA.

2 Contents

3 Capabilities and limitations

3.1 Capabilities and limitations of Oval

The current version of Oval has the following capabilities and limitations:

1. It can perform discrete element analysis on assemblies containing any one of the following particle types: circles (2D), ellipses (2D), ovals (2D), spheres (3D) and non-spherical "ovoids" (3D).

2. At present, it works best with flat periodic boundaries on all sides of a two- or three-dimensional assembly. For two-dimensional assemblies, it can also use flexible boundaries; with both two- or three-dimensional assemblies, it can use rigid boundaries.

3. It can only analyze assemblies with parallel boundaries (i.e. assemblies of 2D parallelogram and 3D parallelepiped shapes).

4. With 2D assemblies, it can create output files for use with the OvalPlot graphics program.

5. It can create output files of both macro-results and micro-results. 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. 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 8.1.14 and 8.1.15). The frictional slider can be "turned off" (Sections 8.1.16 and 8.1.17).

7. It also includes a Hertz-Mindlin contact mechanism, in which the shear modulus, Poissons ratio, and friction coefficient must be specified.

8. 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).

9. It includes the following types of damping: translational mass (global) damping, rotational mass (global) damping, and contact damping (refer to Cundall and Strack 1979). Each may be independently specified or "turned off" (Sections 8.1.20–8.1.22).

10. 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.

11. 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.

12. It can assign initial random velocities to the particles in the assembly (Sections 8.1.23 and 8.1.24).

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

14. It does not yet include gravity effects.

15. The code is not yet parallelized.

3.2 Capabilities and limitations of OvalPlot

The current version of OvalPlot has the following capabilities and limitations:

1. It provides a number of plot types for visualizing an assembly's particle arrangement, topology, particle movements and rotations, micro-deformations within void cells, contact forces, inter-particle movements, etc. (Section 20). There are currently 14 types of plots, and more are planned (Section 20).

2. It can only accommodate 2D assemblies.

3. It can only accommodate data input in a form that is generated by the DEM program Oval.

4. At present, it only produces output in LaTeX format. (This format can be converted Postscript, encapsulated postscript, and other graphics formats.)

5. It can plot either the actual changes in an assembly (movements, rotations, micro-deformations, etc.) or changes relative to the mean assembly deformation (Section 21.2).

6. It can plot the total movements and forces that occur at particle contacts, or it can plot either the tangential or normal components of these movements and rotations (Section 21.3).

7. It can apply a number of "filters" to the contact forces and movements (Section 21.4). For example, it can plot the contributions of individual contact forces to the mean stress of the assembly (Section 20.4). It can plot either the contribution in magnitude or in direction (Section 21.5).

8. It can separately plot the elastic and inelastic particle movements during a loading/unloading cycle (Sections 16.15 and 17).

9. The output plots can be scaled to a particular size or scaled to fit on USletter or A4 paper (Sections 16.4 and 16.18).

10. It can provide headers and footers to the printed page for easier referencing (Sections 16.5 and 19.1).

11. It can provide a legend that gives the length scale of the plot (Sections 16.6 and 19.2).

12. It can provide a legend that gives the scale of intensities for the plotted quantities (Sections 16.7 and 19.3).

13. It can plot a subset of the assembly's particles (Sections 16.16 and 16.19).

14. It allows adjustments in the font size (Sections 16.11 and 16.12).

15. It can plot tiled multiples of assemblies with periodic boundaries (Section 16.14).

16. It allows adjustment of the visual intensities of the plotted quantities (Section 16.17).

3.3 Things to do

3.3.1 Things to do: Oval

The current version of Oval lacks certain capabilities that are currently under development. Most of these capabilities have already been completed for fully circular and spherical particles, but Oval may not yet fully accommodate the recent addition of the non-circular and non-spherical particle types. A partial list of things to do is as follows:

- add code for analyzing the local deformations within the void cells between particles.

- add code for analyzing strain gradient dependence in granular materials.

- add code to "turn on" gravity for assemblies that do not have periodic boundaries.

- make available binaries for other platforms and operating systems on the web site.

Many of these capabilities are currently represented by empty subroutines that simply serve as "place holders" for the final code.

If you plan to use Oval, then please send an email message to the address on the front page, so that you can be kept informed of any upgrades and corrections to the package.

3.3.2 Things to do: OvalPlot

The following additions to OvalPlot are currently planned:

- add code to produce SVG output. OvalPlot is very cumbersome to use. With XML-based SVG output, the files could be displayed immediately with a suitably equipped file browser, such as Firefox, without having to compile Latex code. SVG graphics can also be edited within Adobe Illustrator, Inkscape, etc.
- add code to plot contact forces, contact deformations, and micro-deformations for ellipse and oval particles.
- add code for producing "contact dislocation lines", as described by Murakami, Sakaguchi, and Hasegawa (1997).
- add code for plotting particle rotations and rotation gradients in the vicinity of individual void cells.
- add code to plot the separate effects of contact rolling and sliding.

If you plan to use OvalPlot, then please send an email message to the address on the front page, so that you can be kept informed of any upgrades and corrections to the package.

3.4 How you can help

You can help in the development of Oval and OvalPlot in a number of ways:

- send bug reports to the email address on the first page.
- send binaries for other platforms and operating systems.
- send your own StartFiles of initial particle arrangements. It would be nice to have a catalog of particle arrangements with various shape distributions, size distributions, densities, and anisotropies.
- develop drivers for other graphics formats.
- add an index to this document?

4 Availability and installation

The program is written in a rather inelegant dialect of Fortran 77. Source code is freely available under terms of the open source GNU General Public License (GPL), version 2. The entire program can be downloaded from the author's web repository at

`http://faculty.up.edu/kuhn/oval`

including the source code, executable binaries, this documentation, sample assemblies, and other example files. To install the program you should follow the steps that are described below.

4.1 Unix systems

1. Download `oval-latest.tar.gz` from the web site. This file is a symbolic link to the latest version of the software, although some earlier versions may also be archived at the site.

2. Send an email message to the address on the front page of this documentation, so that I can keep you informed of any upgrades and corrections to the OVAL package.

3. Create a directory for the program. Place the `oval-latest.tar.gz` file into the directory, and then uncompress the file:

   ```
   gunzip oval-latest.tar.gz
   ```

   The directory should now contain the archive file `oval-latest.tar`.

4. Extract the contents of the file:

   ```
   tar xvf gunzip oval-latest.tar
   ```

   This command will create several new subdirectories and unpack the contents into those directories:

   `source` the Fortran source files, including the following files:
   `oval-X.X.XX.f`
   `common-X.X.XX`
   `ovalplot-X.X.XX.f`
   `common-plot-X.X.XX.f`
   `dsort.f`
   `texdraw_oval.tex`

`bin` executable binary files. These files include the author's i586 Linux and Windows binaries and any other contributed binaries.

`doc` this and other documents in various formats.

`samples` sample assemblies and example input files for both OVAL and OVALPLOT.

I recommend that you write-protect all of these files to prevent their corruption, since you will likely want to use them as templates for your own code changes and simulations.

5. If you are using Linux on an i586 machine, you can use the author's executable binary files `oval` and `ovalplot`, which are found in the directory `bin/linux_i586`. Otherwise, you may need to compile the program for your own platform or operating system. This will certainly be necessary if you modify the source code for your own special purposes. The program has, at various stages in its development, been compiled on the following platforms: a DEC MicroVAX workstation, a DEC VAX750 mainframe, a Data General mainframe, Sun SPARC ipx workstations, a Pentium III PC (Linux Caldera 2.3, Suse 6.3, and Mandrake 7.2), and an Athlon PC (Mandrake 8.1). Recent development has been primarily on Linux platforms using, the GNU `g77`, Portland Group `pgf77`, and Intel `ifc` Fortran compilers.

6. If necessary, compile OVAL and OVALPLOT as you would compile any Fortran program, using your resident compiler while in the appropriate directory:

```
f77
g77
pgf77 -o oval {your options here} oval-X.X.XX.f
etc77
```

and

```
f77 -o ovalplot {your options here} ovalplot-X.X.XX.f
```

where `X.X.XX` is the appropriate version of the Fortran source code. With these commands, the output binary (executable) files will be named `oval` and `ovalplot`. The following five files must reside in the same directory for the compile to work:

```
oval-X.X.XX.f
common-X.X.XX
ovalplot-X.X.XX.f
```

```
common-plot-X.X.XX
dsort.f
```

The file `dsort.f` contains the quick-sort subroutine `dsort` written by R. E. Jones and J. A. Wisniewski as part of Sandia Laboratory's SLATEC repository. It is separately licensed.

You may wish to consider your own options when compiling the source Fortran code. For example, you may want to try improving the precision by using something like a "`-r8`" option, which forces the use of double precision, 8-byte floating point numbers. For very large assemblies, you may need to use something like "`-i4`" to force the use of 4-byte integers for indexing the many particles and contacts (you will also need to change the value of the parameter "`mp`" in the `common` file). Some compilers offer options for optimizing the binaries (perhaps "`-fast`" or "`-O`") and for producing the extra information required by debuggers (perhaps "`-g`"). Consult your compiler's manual for the actual usage.

There are several aspects of the `oval-X.X.XX.f` code that are not true Fortran 77 and many lead to compiling errors or warnings. These include the following:

- the use of the compiler-local "`include`" statement that inserts code from the files `common-X.X.XX.f` and `commonplot-X.X.XX.f` into the bodies of the `oval-X.X.XX.f` and `ovalplot-X.X.XX.f` code.
- the occasional use of "`do while`" loops
- use of the compiler-local functions `rand` and `srand` for creating and seeding random number sequences.

These variances are accommodated by both the g77 and pgf77 compilers.

7. You may want to place the executable files `oval` and `ovalplot` into an appropriate `bin` directory on your system's search `$PATH`. You may also want to use symbolic links to your entire `oval` directory or to specific files and subdirectories.

8. Several utility programs will help in using Oval and OvalPlot, and these utilities are described in Section 5. All of these utilities are standard on most Unix/Linux distributions.

4.2 Windows systems

1. Download the file `oval-latest.zip` from the web site (page 12). This file is a symbolic link to the latest version of the software, although some earlier versions may also be archived at the site.

2. Create a directory (folder) for the program files. Unzip the file

   ```
   oval-latest.zip
   ```

   using `PKUNZIP` or a similar utility, placing its contents into the new directory. This will both extract and uncompress the files. The directory should now include a number of new subdirectories, which were listed and discussed in item 4 on page 12.

3. You can use the author's binaries `oval` and `ovalplot` in the `bin/windows` directory. If you make any changes to the program source files, you will need to recompile the program to create your own executable files. The procedure is described in item 5, page 13.

4. Several utility programs will help in using Oval OvalPlot, and these utilities are described in Section 5. Without these utilities, you will probably have difficulties in running Oval and creating the graphics plots.

5 Helpful/essential utilities

Besides the binaries for Oval and OvalPlot, a number of utilities will be useful (and, in some cases, essential). All of these utilities are standard on Linux systems, so unless your Linux installation is of the barest extent, these utilities should already be available. For Windows systems, you will likely need to download at least some of these utilities, although all are available in freeware, shareware, or commercial forms. The various utilities are listed in Table 1 and discussed below.

1. a text editor. To use Oval and OvalPlot, you will need a text editor for creating the input files and reading the output files. I would suggest that Microsoft Word and WordPad be avoided as text editors, because of the residue "line feed" characters that they can embed in your text files. Windows users will want to search the web for a real text editor. (You can try `http:www.zdnet.com` and search for "text editor", which will lead you to perhaps hundreds of downloadable candidates, complete with reviews.) The editor should be capable of accommodating widths of 112 columns, as this is the size of certain

Item	Description
1	text editor
2	spreadsheet
3	Fortran compiler and debugger
4	LaTeXpackage
5	dvi viewer
6	ps viewer and print utility
7	Postscript-to-pdf converter
8	pdf viewer and print utility
9	ps-to-eps converter
10	data analysis software

Table 1: Helpful and essential utilities

input files. Your Windows Fortran compiler may already include a text editor or shell environment.

2. spreadsheet software such as Excel (commercial) or Gnumeric and OpenOffice.org (open source).

3. a Fortran compiler and debugger. Both the open source g77/gdb and robust commercial compilers and debuggers are available for Unix systems. On Windows, commercial compilers are available. Also consider the Windows porting of the gcc/g77/gdb utilities. This package, named djgpp, can be downloaded as a set of zip-files from

   ```
   http://www.delorie.com/djgpp/
   ```

   (thanks Miklós!).

   The only problem with the free gdb debugger is that it may not support the use of Fortran `common` statements, which are liberally used in the Oval and OvalPlot packages. The 3.0 version of gdb, currently available for Unix/Linux, supposedly supports `common` statements, but I haven't tested this yet.

4. the LaTeX package for document preparation, including the following standard style files

   ```
   fancyheadings.sty
   texdraw.sty
   texdraw.tex
   txdps.tex
   ```

and the author's set of custom macros for color extensions to the texdraw package:

```
texdraw_oval.tex
```

You will be unable to display your OvalPlot graphics without LaTeX and these supplementary files. You will probably also need the dvi-to-postscript converter `dvips`.

The entire LaTeX package is standard with nearly all Linux distributions (or the prepackaged teTeX LaTeX package can be downloaded in tar, rpm, or dbm formats). On Windows systems, you will want to download and install the MiKTeX package, which includes LaTeX and the most common utilities. For downloads of MiKTeX, start your search at the web site `http://www.ctan.org`.

5. a dvi file viewer. These viewers are standard on Linux systems (xdvi, kdvi, etc., or download as part of the teTeX package. I have found that xdvi more faithfully renders the images) and are part of the MiKTeX package for Windows.

6. a postscript viewer and print utility. These are also standard on Linux systems (ghostview, gv, ggv, etc.) The Ghostscript and GSview packages are freely available for Windows from the following web site:

   ```
   http://www.cs.wisc.edu/~ghost/
   ```

   The commercial Adobe Acrobat suite also contains a postscript viewer.

7. a postscript-to-pdf converter. The standard Linux utility ps2pdf will convert your postscript files to a pdf format. On Windows systems, the commercial Adobe Distiller will also perform this task.

8. a pdf viewer. The freely available Adobe Acrobat Reader will serve this purpose, as will the Linux utility xpdf.

9. a ps-to-eps converter. Postscript (ps) graphics can be embedded in a document by first converting the postscript (ps) file into an encapsulated postscript (eps) format. The Linux utility psfixbb does a nice job of this. Refer to the following web site:

   ```
   http://www.strw.leidenuniv.nl/~dominik/Tools/psfixbb/
   ```

   The command structure is

   ```
   psfixbb -o <file_name>.exp <file_name>.ps
   ```

If it doesn't work the first time, try changing the first line of `psfixbb` to give your perl location (try the command: `which perl`).

10. data analysis software. If you want to perform your own analysis of the micro-level F-files, you will probably wand to use an analysis package such as Matlab, Octave, Scilab, or R.

6 Running Oval

Oval is *not* yet an interactive program with a graphical user interface. At present, it runs only in a batch mode. Sections 15 and 17 discuss the integration and running of Oval and its graphics post-processor OvalPlot. In the current section, we consider only the running of Oval.

You would normally run Oval from within a terminal (e.g. an xterm window or DOS console) with the following command at the system prompt:

```
<path>oval
```

where `<path>` is the path to your executable `oval` file. Instead of explicitly providing the `<path>`, you may wish to create a link (shortcut) to the `oval` binary, place it in a directory in your system's search `$PATH`, or modify your system's search `$PATH` to include the directory that contains `oval`.

You will then be queried for the names of two files,

```
Name of the RunFile:
```

and

```
Name of the StartFile:
```

We will henceforth refer to these two files with the generic names "RunFile" and "StartFile", which are described in Sections 8 and 9. 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 2 and will be described later in the documentation. As a simple example, the output file `A<RunFile>.txt` is a text file that can be imported into a spreadsheet, such as Excel or Gnumeric to view the average stresses and deformations of the assembly (note, *imported*, not opened).

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 2).

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
E<RunFile>	binary	StartFile	8.1.7
F[1-4]?<RunFile>	text	micro-results for post-analyses	10.5
G?<RunFile>	binary	graphics input for OvalPlot	8.2.13
? – a letter that corresponds to the deformation-stress path in which the file was created (Section 8.2.7 and Section 17)			

Table 2: 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 boundaries 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 8.1.17). 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 8.1.17). 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 8.1.12). For example, if `nplatn=4`, then you will be querried 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 contrained 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 postion 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 OVAL web site (see page 12) 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 2). 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 15 and in item 9 on page 65).

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 31. 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:

```
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 contact stiffness (force/indentation)
    1.00         : kratio   | ratio of tangential/normal contact stiffnesses
    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.           : rfree1   | unused input value (0) A placeholder for added features
    0.           : rfree2   | unused input value (0) A placeholder for added features
    0.           : rfree3   | unused input value (0) A placeholder for added features
    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.

`algori=1` The conventional DEM algorithm (refer to Cundall and Strack 1979). The algorithm uses an implicit integration scheme.

`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 `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 perserving backward-compatability 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, 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 8.1.24), 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 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 10 and 500 should be fine, but larger values will lead to somewhat faster computations. See Section 8.1.19.

8.1.10 icirct (i16)

Currently not supported.

8.1.11 imodel (i16)

The contact model.

`imodel=0` A linear contact model with friction (see Sections 8.1.14, 8.1.15, and 8.1.16).

`imodel=5` A Hertz-Mindlin contact model with friction. (see Sections 8.1.14, 8.1.15, and 8.1.16).

8.1.12 `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`.

8.1.13 `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`.

8.1.14 `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 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 is `kn`/2, so that the OVAL stiffness value may be half of that used in other DEM codes.

With Hertz-Mindlin contacts (`imodel=5`), 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.

8.1.15 `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 (`imodel=5`), this input variable is the Poisson ratio of the particle material.

8.1.16 `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.

8.1.17 `frictw` (f16.7)

The friction coefficient between particles and boundary walls (or boundary particles). See Section 7.

8.1.18 `rho` (f16.7)

The mass density of the particle material. See Section 8.1.25 for options to automatically assign a value of `rho`.

8.1.19 `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 8.1.9). 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.

8.1.20 `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 47–48).

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

A dimensionless coefficient of viscosity, applied to the rotational velocities of the particles (see the previous Section 8.1.20).

8.1.22 `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.

8.1.23 `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 8.1.24.

8.1.24 `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).

8.1.25 `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 8.1.18):

`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 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. 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 $\dot{F}_{23}$. 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 (see the third matrix above).

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 or 1. When the digit is 0, the deformation threshold is monitored; when the digit is 1, the 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 ($10 \times 0.250 = 2.500$).

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 Bardet (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 creat rigid-flat boundaries on the two x_1 faces; and an `iflexc=90` will creat 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 17 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. 3.

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

4. an angle β that describes the manner in which circular arcs are spliced together to create oval and ovoid particles (Fig. 4). 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

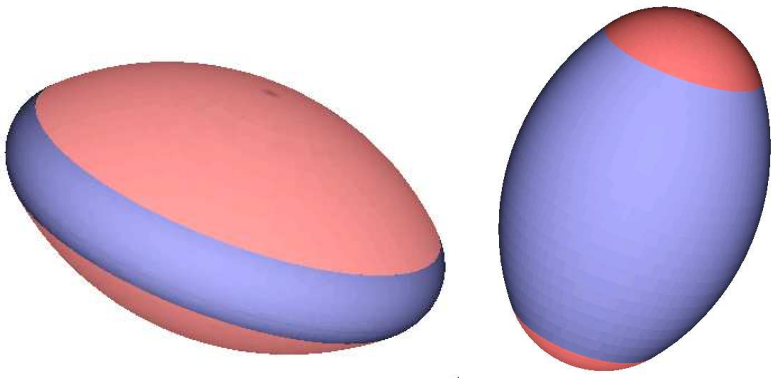
```
1
  1002  2.98994563276730430E+01  2.98446889993219070E+01  1.00000000000000000E+00
        0.00000000000000000E+00  0.00000000000000000E+00  0.00000000000000000E+00
  3.13454063710418570E-01  1.79352868741551070E+01  1.79266777574543750E+01
  4.43756654693070110E-01  1.27698119609280810E+01  2.52760548238972190E+01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



(a) Oblate (b) Prolate

Figure 2: Ovoid particles.

`kshape=1` circular (2D) disks

`kshape=2` oval (2D) disks. The ovals are composed of four circular arcs spliced together (Fig. 4). 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.

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. 3). 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).

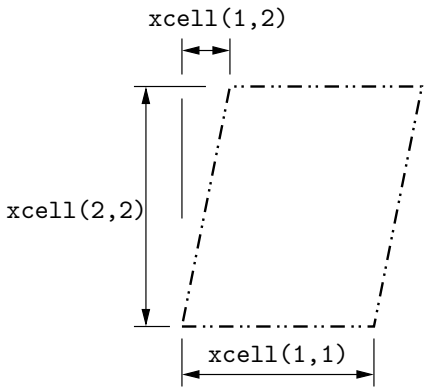


Figure 3: The `xcell(i,j)` dimensions for a 2D assembly

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. 3).

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. 4). 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:

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

9.2 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 on 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 line between the fields. In Oval versions 0.6.0 and higher, the data can be

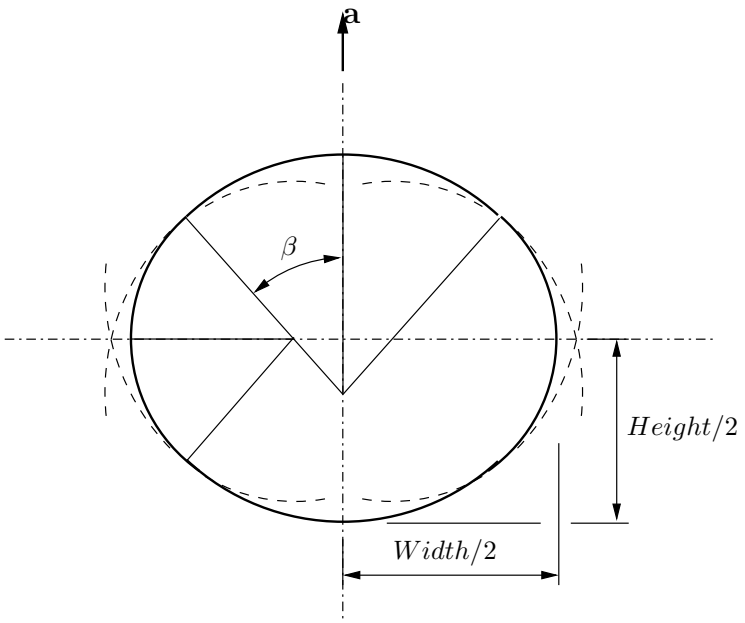


Figure 4: Geometry of an oval composed of four circular arcs.

provided in *free format*, so that data fields do *not* have to be aligned within certain columns. The example input on page 37 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. 5). The location coordinates refer to the center of the particle. Following the general information at the head of a DStartFile, this information is given on 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

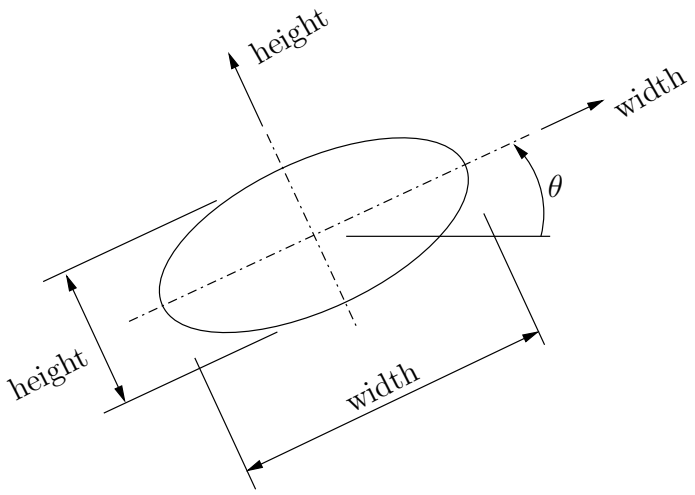


Figure 5: Orientation angle θ for elongated particles (2D ovals and ellipses).

height). The width is measured as shown in (Fig. 5), 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. 5). 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. 5).

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. 5). The location coordinates refer to the center of the ellipse. Following the general information at the head of a `DStartFile`, this information is given on 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. 5). 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. 5).

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 on 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. 4) 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 on 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**.

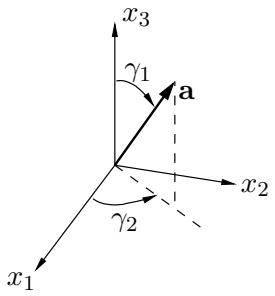


Figure 6: Orientation angles γ_1 and γ_2 for 3D ovoid particles. Vector **a** is the central axis of revolution of the ovoid (Figs. 4 and 5).

- 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. 6). 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. 6).

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 (Satake 1992; Satake 1993; Kuhn 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 (Satake 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 8.1.25).

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 44. 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 interia 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. 3.

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 5), 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 8.1.13). 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):

```
 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 3). 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 3) 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

	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 3: Contents of the various F-files. Note that the "?" character in a file name (Table 3) 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).

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 3), 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. 3, page 39. The `def(i,j)` deformations are components of the deformation gradient **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, 8.1.14, 8.1.15, and 8.1.16.

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. 4, page 40)
Field 3	Aspect ratio, Height/Width (Fig. 4, page 40)
Field 4	x_1 position
Field 5	x_2 position
Field 6	θ orientation in radians (Fig. 5, page 41)

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

These fields contain information on every contact within the assembly. 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 Preparata and Shamos (1985), which describes the DCEL and how to use it to navigate the topology of a 2D planar graph. As an example, Table 4 shows five rows and the first six columns in an Fc-file (Fig. 7).

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 4: An example DCEL table (Fig. 7).

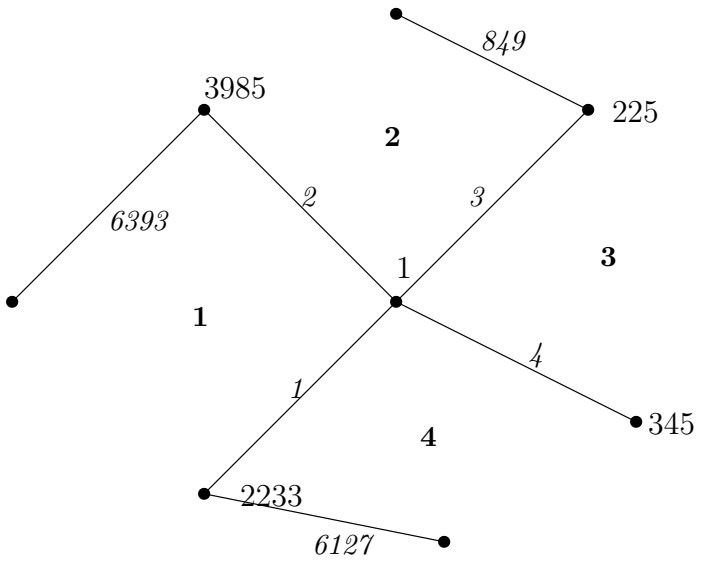


Figure 7: The particle graph associated with the DCEL of Table 4. The dots represent particle centers. Lines represent the contacts between particles.

- Row 3 in an F-file corresponds to contact 3 (see row 3 in Table 4).
- 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 20.2 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 (Satake 1992). The `hf` data in an Fd-file points to the starting (header) contact for a void cell (Section 10.6.4).

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 8.1.11). See (Thornton and Randall 1988).

10.6.4 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).

10.7 F-files for 3D assemblies

Three F-files are created at once (Table 3), and they are described in the following three subsections.

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. 6, Section 9.2.5)
Field 7	γ_2 orientation angle (in degrees) of the particle's axis, (Fig. 6, 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)

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 8.1.11). See (Thornton and Randall 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 8.1.11). See (Thornton and Randall 1988).

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 performed (see Section 8.1.19). The integer `ipt2` is the current length of the near-neighbor linked list (Section 8.1.19). 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 12. The sample StartFiles

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.

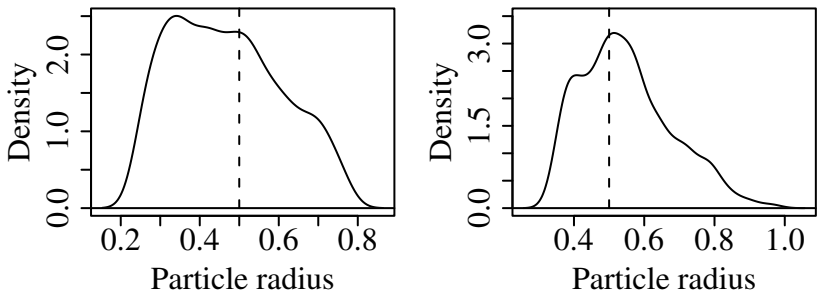
Table 5: Attributes of several sample assemblies

should be located in your `oval` directory in

```
samples/startfiles
```

The primary attributes of these assemblies are shown in Table 5. Each assembly is roughly square (or cubical) and with an isotropic fabric. They were created by isotropically compressing a sparse assembly with friction 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. 8. The histogram is centered on the median size $0.50D_{50}$. In Fig. 8a, the radii of non-circular 2D particles refers to their mean radii, (Height + Width)/4 (Fig. 5, Section 9.2). In Fig. 8b, the radii of non-spherical 3D particles refers to their mean radii,



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

Figure 8: Histograms of particle radii for 2D and 3D assemblies

(Height + 2 · Width)/6 The distribution of aspect ratios for oval and ovoid assemblies are described in the footnotes of Table 5.

The void ratio is a measure of the packing density of a granular assembly (Table 5). 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 5 are computed as twice the number of contacts divided by the number of particles.

Among the attributes listed in Table 5, 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 5, the website includes a directory `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 5 and Fig. 8a. 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.

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 6: Execution times for simulations with the RunFile shown in Fig. 1, page 24 and various StartFile assemblies (Table 5). The times are with an Intel Pentium III 450MHz processor.

13 Example simulations using Oval

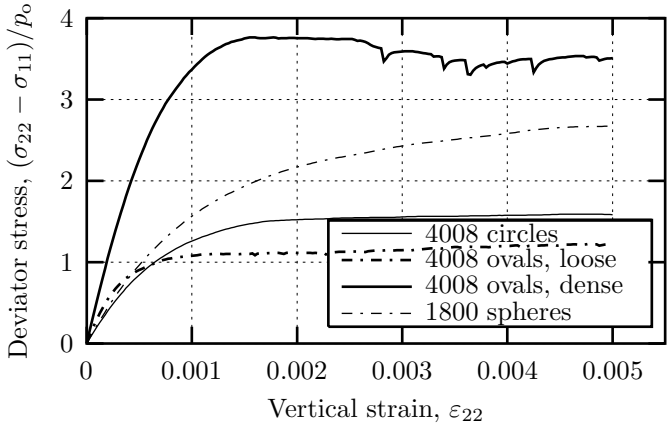
As an example simulation, consider the RunFile named `LoadComp` shown on page 24. After a brief initial period in which the assembly is allowed to equilibrate, the assembly is vertically compressed while maintaining constant horizontal stress ($\dot{F}_{22} < 0$, $\dot{F}_{11} = 0$). For 3D assemblies, the deformations were under plane strain conditions ($\dot{F}_{33} = 0$). The results for all of the larger assemblies are shown in Fig. 9 and are archived at the web site (page 12). These results can also be found in the your `oval` directory in

```
samples/results
```

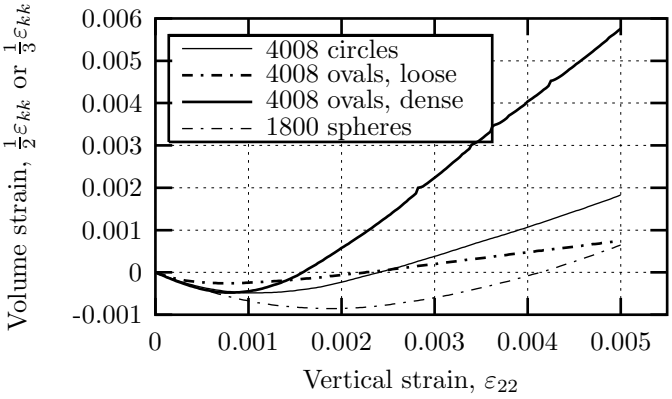
The deviator stress in Fig. 9 has been normalized by dividing by the initial mean stress, p_o. Note the quite large difference in strengths of the loose and dense assemblies of ovals. The loose assembly (created quite by accident) is much weaker than the assembly of circular discs, even though the two 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 6, 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 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.



(a)



(b)

Figure 9: Results for both 2D and 3D materials: deviator stress and volumetric behavior.

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 8.1.19). 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 5, page 59).

   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 58), 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.

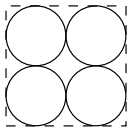
5. Binary E-file and C-file formats might not be portable between different platforms or operating systems.

6. If you are assigning initial random velocities to the particles (an input value `rmsvel`$\neq$`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 8.1.12).

15 The OvalPlot process

Like its partner Oval, OvalPlot is not an interactive program with a refined user interface. You would normally run OvalPlot and its various utility programs from within a terminal (e.g. an xterm window or DOS console). The complete process, from the running of Oval to the printing of the graphical output, is shown schematically in Fig. 10. In this figure,

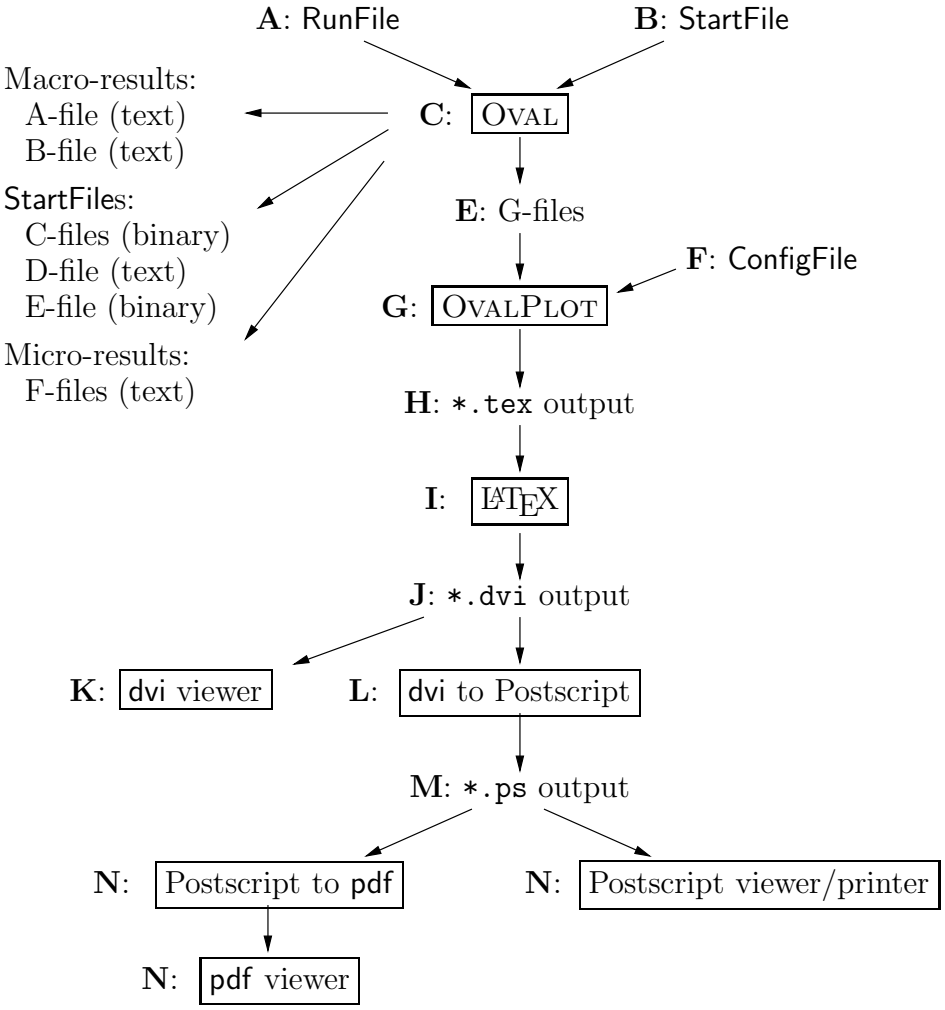


Figure 10: The OvalPlot process.

the solid boxes designate executable processes. The phantom boxes show the input and output files of these processes.

The steps that are shown in Fig. 10 are described below. Further details are provided throughout this document. You should also consult Section 14, "Some advice."

A. Create a RunFile, which will describe the manner and sequencing of the DEM simulation (Section 8). Samples can be found in the directory

```
oval/samples/runfiles
```

B. Create a StartFile, which gives the initial arrangement of the particles. There are three forms of StartFiles, and all are described in Sections 8.1.7, 8.1.8, 8.2.7, and 9. Although it is possible to write your own StartFiles for simple, regular particle arrangements, creating irregular packings of multi-sized particles is quite involved. You may want to use the sample StartFiles that are provided with the OVAL package (in your directory `samples/startfiles`). I plan to describe my own methods for producing such packings in a future revision of this documentation, but I would also like to include descriptions of your own methods and include your own StartFiles as samples in the OVAL package. Please send them along with a brief description!

   Note that StartFiles can be created as output files during the running of OVAL. All of the sample StartFiles were created by running and rerunning OVAL, using the output from on simulation as the input to another, until the final assembly was formed.

C. Run OVAL using the input from steps A and B. This process is described in Section 6.

D. The output from OVAL can take many forms (see Table 2, page 19, for a summary). Besides creating new StartFiles, the output can include summaries of the macro-level results (A-files and B-files, Sections 10.1–10.4). and micro-level results (F-files, Sections 10.5–10.7). A-files can be imported into a spreadsheet (Section 10.1), B-files can be read with a text editor (Section 10.2), and F-files are intended for use with data analysis packages such as Matlab, Scilab, R, etc. Note that the C, D, and E StartFiles can be used as input to subsequent OVAL simulations (step C).

E. The output from OVAL can also include G-files, which are the binary input files that will be used for creating graphics plots (see Sections 8.2.13, 17, and 18).

F. Create a ConfigFile for configuring the graphics output (Section 16). You should also create the `path` directory that is given in the ConfigFile. Place a copy of the file `texdraw_oval.tex` into the new directory.

G. Run OvalPlot, using the output from steps E and F. Your ConfigFile should reside in the same directory from which OvalPlot is run. See Section 18.

H. The output from OvalPlot is a Latex file with embedded graphics formatting. The name of the output Latex *.tex file will appear on the screen while running OvalPlot. For example:

```
Output file: results/dummy_cell_def.tex
```

This file will require several steps of processing before the results can be realized in a displayed or printed form.

I. Run Latex:

```
latex <file name>.tex
```

using the file name that was displayed while running OvalPlot (step G). You will need to install the LaTeX software package before performing this step (see Section 5, page 17).

You may need to change into the directory that contains the file `<file name>.tex` before performing this (and subsequent) steps.

The directory within which you run OvalPlot must contain the special `texdraw_oval.tex` macro file (item 4 in Section 4.1). **Without this file, Latex will not run.**

J. The output from Latex will include a binary `*.dvi` file and several postscript fragment files (e.g. `*.ps1`, `*.ps2`, etc.).

K. You can display the graphical output with a dvi viewer (Section 5, page 17), or ...

L. You can run dvips to produce a postscript file:

```
dvips -o <file name>.ps <file name>
```

M. You will now have a postscript file for producing the printed output with Ghostscript or similar software (item 6 in Section 5).

N. You can view and print the postscript file with the Ghostview viewer or Adobe Acrobat Distiller. You can also convert the postscript file to pdf format for viewing and printing with Ghostscript, xpdf, or the Adobe Acrobat package (items 6–8 in Section 5).

16 The ConfigFile

The ConfigFile describes the style of graphical presentation. An example is given in Fig. 11, and a sample ConfigFile can be found in your main `oval` directory in `samples/plot`. Before running OvalPlot, you should place a copy of your own ConfigFile in the same directory from which OvalPlot is run. You can then customize your ConfigFile as needed.

A ConfigFile consists of the following four parts:

- a single title line (Section 16.1)
- a single path line (Section 16.2)
- a series of 16 formatted lines that provide integer input (Sections 16.3 to 16.16)
- a series of 7 formatted lines that provide floating point data (Sections 16.17 to 16.19))

The format of these 25 lines is

```
a80,/,a80,15(/,i16),7(/,f16.7)
```

and each data item is detailed in the sections below.

16.1 `title` (a80)

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

16.2 `path` (a80)

This gives the directory path (relative to the current path) for placing the output from OvalPlot. Eight or more files can be generated for each plot, so you will probably want to place this output into a separate directory. It is important, however, that the directory (path) given by the input value `path` actually exists. Create this directory before running OvalPlot.

The `path` name should begin in the first column of the line: don't put any blank spaces before the `path` name.

You must place a copy of the file `texdraw_oval.tex` into the new directory (see page 12). A copy of this file can be found under your `oval` directory in `sources` (see page 12).

```
Prototype ConfigFile for the DEM plotting program OvalPlot
results          : apath    | path of the output directory
       1         : iform    | format of output               latex graphics=1
       1         : ipaper   | paper size        0=special 1=USletter 2=EuroA4
       1         : iheadr   | include headers and footers ?        0=no 1=yes
       1         : iscal1   | include a length scale bar ?         0=no 1=yes
       1         : iscal2   | include an intensity scale bar?      0=no 1=yes
       1         : icircl   | circles at the origins of arrows?    0=no 1=yes
       1         : iarrow   | an arrow at the terminus of arrows?  0=no 1=yes
       0         : ilabel   | labels for particles and void cells? 0=no 1=yes
      12         : ifont    | base font size (Latex choices: 10, 11, or 12 pt)
       3         : jfont    | font size (Latex: 0=tiny -> 4=normal -> 9=Huge)
       2         : istran   | reported strain: 0=none, 1=(11), 2=(22), 3=(12)
       1         : ncopy(1) | wallpaper copies in the x_1 direction
       1         : ncopy(2) | wallpaper copies in the x_2 direction
       0         : iplast   | plots of elastic/inelastic rates?    0=no 1=yes
       0         : isub     | plot a subset of the assembly?       0=no 1=yes
      1.         : vivid    | alter the default vividness of the plots
   1000.         : mheigh   | max. height of the plot (when ipaper=0)
   1000.         : mwidth   | max. width  of the plot (when ipaper=0)
      0.2        : x1min    | min. x1 of a subset (when isub=1)
      0.4        : x1max    | max. x1 of a subset (when isub=1)
      0.2        : x2min    | min. x2 of a subset (when isub=1)
      0.6        : x2max    | max. x2 of a subset (when isub=1)
      0.         : sclwid   | width of the color scale bar (0.=default)
```

Figure 11: An example ConfigFile.

16.3 iform (i16)

The form of the output:

`iform=0` This option is not currently available

`iform=1` Latex graphics will be produced. See Section 15 and Fig. 15, which discusses the procedure for processing the Latex output so that the graphics can be displayed on the screen or sent to a printer.

16.4 ipaper (i16)

The paper size for the graphics output.

`ipaper=0` A user-specified size. When `ipaper=0`, the ConfigFile input `mheigh` and `mwidth` will determine the actual height and width of the graphics (Section 16.18). This option is useful when you wish to scale the graphics to a specific size (for example, for an embedded graphic in a report, slide, etc.)

`ipaper=1` The graphics will be scaled to fit on US Letter paper, leaving room for margins.

`ipaper=2` The graphics will be scaled to fit on A4 paper, leaving room for margins.

16.5 iheadr (i16)

If desired, OvalPlot will produce headers and footers on the graphics page. See Section 19.1 for the contents of headers and footers.

`iheadr=0` No headers or footers. I use this option when the graphics will be imported into a report as a figure, and headers and footers must be stripped from the graphics.

`iheadr=1` Include headers and footers (Section 19.1).

16.6 iscal1 (i16)

OvalPlot can include a bar that provides a length scale for the plot. See Section 19.2 for a description of the length scale bar.

`iscal1=0` No length scale bar.

`iscal1=1` Include a length scale bar.

16.7 iscal2 (i16)

OvalPlot can include a scale that shows the intensity of movements, stress, force, rotation, etc. The various intensity scales are described in Section 19.3.

`iscal2=0` No intensity scale.

`iscal2=1` Include an intensity scale.

16.8 icircl (i16)

When vectors are plotted, a small circle can be placed at the start of each vector.

`icircl=0` No circles.

`icircl=1` Include circles.

16.9 iarrow (i16)

When vectors are plotted, an arrowhead can be placed at the end of each vector.

`iarrow=0` No arrowheads.

`iarrow=1` Include arrowheads.

16.10 ilabel (i16)

OvalPlot can place numeric labels on the particles and void cells. Not all plotting formats are supported (see Table 7, page 82).

`ilabel=0` No labels.

`ilabel=1` Include labels. You may also want to adjust the font sizes to avoid a forest of overlapping labels (see Sections 16.11 and 16.12).

16.11 ifont (i16)

The Latex base font size. With Latex, a base font size is selected, usually for an entire document, and then variations are made relative to the base size (Section 16.12). *There are only three base font sizes.*

`ifont=10` A 10 point base font.

`ifont=11` A 11 point base font.

`ifont=12` A 12 point base font.

16.12 jfont (i16)

The Latex font size relative to the base font size. For example to produce the smallest possible fonts (which I will use when placing labels on each particle, Section 16.10), you would combine `ifont=10` and `jfont=0`. The largest possible font is produced with `ifont=12` and `jfont=9`.

`jfont=`	`0`	tiny	`5`	large
	`1`	scriptsize	`6`	Large
	`2`	footnotesize	`7`	LARGE
	`3`	small	`8`	huge
	`4`	normalsize	`9`	Huge

16.13 istran (i16)

OvalPlot can report the strain level of the plot in the upper right header (Sections 19.1 and 16.5). This feature can be used to help identify the origin of individual plots among stacks of similar plots. OvalPlot has no way, however, of guessing the particular strain component that would be most useful for you. As an example, if you are running shear tests, `istran=3` would be most appropriate.

`istran=1` Report the 11 strain component.

`istran=2` Report the 22 strain component.

`istran=3` Report the 12 strain component.

The strains reported by Oval and OvalPlot are a bit unusual. See Sections 10.1 and 10.3.2 for a description.

The input value `istran` is also used in computing the elastic component of deformation (see page 78).

16.14 ncopy(1), ncopy(2) (i16)

This feature allows you to "tile" multiple copies of an assembly with periodic boundaries. The values of both `ncopy(1)` and `ncopy(2)` are normally set to one, so that only a single assembly is displayed in the plot. When, for example, `ncopy(1)=3`, three assemblies are tiled side-by-side. Figure 17a (page 97) shows an example with `ncopy(1)=2` and `ncopy(2)=1`. When `ncopy(2)` is greater than one, the assembly is also tiled vertically.

16.15 `iplast` (i16)

OvalPlot can be used for separately displaying the elastic (recoverable) and inelastic (permanent) deformations and movements. See Section 17 for information on creating the necessary G-file input.

`iplast=0` Do not separate the results into elastic or inelastic contributions. Instead, plot the total movements and deformations. This option requires, at most, two G-files (Section 17).

`iplast=1` Calculate the separate elastic or inelastic contributions to deformations and movements. When running OvalPlot with this option, you will be queried whether you want to display the elastic, inelastic, or total deformations (Section 21.6). Three G-files will be required (Section 17). Note that this option is not supported with all plot types (Table 7, page 82).

16.16 `isub` (i16)

OvalPlot can isolate and plot a subset of the entire assembly.

`isub=0` Plot the entire assembly.

`isub=1` Plot a subset of the assembly. The subset will only include particles that lie within a designated "window." See Section 16.19 for the manner in which the window is defined within the ConfigFile.

16.17 `vivid` (f16.7)

The graphic plots illustrate the magnitudes of particular micro-quantities by using several graphics strategies: arrow vectors, line thicknesses, colors, color intensities, etc. I have established a default scaling system that automatically adjusts the scaling in each strategy to provide reasonably lucid plots (at least with the sample assemblies that I tested). For example, arrow vectors are used for illustrating particle movements. The default scaling attempts to display these arrows at lengths that are short enough to avoid a dense tangled forest of arrows, but with arrows that are long enough to reveal both patterned and amorphous behaviors (see Fig. 17a).

You can alter this default scaling to perhaps better reveal a particular micro-feature. This alteration is done with the configuration variable `vivid`.

`vivid=1.0` Use the default scaling of colors, vector lengths, line widths, etc.

`vivid`≠`1.0` Scale the default coloration, vector lengths, etc. Values of `vivid` greater than one will have the following effects:

- vector arrow lengths are increased
- line widths become thicker
- colors become *less* intense (more pale)

Vivid must be greater than zero.

16.18 `mheigh, mwidth` (`f16.7`)

When `ipaper=0`, the input values `mheigh` and `mwidth` give the height and width of the graphics plot in inches (sorry!). When `ipaper`≠`0`, the values of `mheigh` and `mwidth` are ignored. The dimensions `mheigh` and `mwidth` are only approximate, and some tweaking may be necessary to get an exact desired size. For example, the true height of a plot depends upon such complexities as inter-line spacings and font size, and these factors are only approximated in the application of `mheigh`.

16.19 `x1min,x1max,x2min,x2max` (`f16.7`)

When `isub=0`, these four values are ignored. When `isub=1`, the four values define the "window" that will be used to isolate and plot a subset of the assembly (Section 16.16). Each input value should be in the range of 0.0 to 1.0. Each value is referenced to either the full height or full width of the assembly. For example, if `x1min=0.333` and `x1max=0.333`, the plot will only include particles whose centers lie within the middle 1/3 of the assembly, and particles outside of the middle 1/3 will be ignored. At present, OvalPlot does not support subsets that straddle periodic boundaries.

16.20 `sclwid` (`f16.7`)

The value of `sclwid` is the approximate width of the intensity scale (Section 19.3). When `sclwid=0.`, then the width will be about half the width of the plot.

17 Creating the G-files

The graphics input to OvalPlot is created by first running a DEM simulation with Oval, which produces the necessary G-files (see Sections 15 and 18). These files can be quite large: for an assembly of 1000 particles, a binary G-file will be about 200kBytes.

- For plots that only show the assembly's status at a particular time, only a single G-file is required. These plots include simple graphics of particle locations, the particle graph (topology), and contact forces (`ktype=1-4`, Sections 20.1 to 20.4 and Table 7, page 82). Before running OVAL, set `iplot=1` in the RunFile to create a G-file at the the beginning of a deformation-stress segment (Section 8.2.13). For example, the following input at the bottom of a RunFile could be used to create a G-file at a vertical strain of 1% ($= 0.2 \times 10^{-5} \times 5000$):

```
     0.          : rfree3   | unused in
     0.250       : dt       | time incr

       **********  Controlling Strain        krotat            iplot
        (100000)   (10000)   (1000)            |                 |
icontr| rate_11 | rate_22 | rate_33 |      igoal   finalv         |
------|---------|---------|---------|      |--|-|---------|     |--|
000000      0.        0.        0.          70 0     2.5          0
100000      0.     -0.2e-5      0.    ...   70 0  5000.0    ...   0
100000      0.     -0.2e-5      0.          70 0     0.25         1
```

- Two G-files are required for plots that depict rates or changes within the assembly: particle velocities, rotation rates, inter-particle movements, deformations, etc. (`ktype=10-23`, Sections 20.5 to 20.14). The output plots will display these rates in a dimensionless form. For example, the velocity of the k^{th} particle is computed as the change in its position between two G-files divided by D_{50} and the change in deformation:

$$(\mathbf{x}^2_{\text{2nd}} - \mathbf{x}^2_{\text{1st}})/(D_{50}|\overline{L}|) \tag{15}$$

  where $|\overline{L}|$ is the tensor norm of the approximate velocity gradient,

$$|\overline{L}| = \left|(\mathbf{F}_{\text{2nd}} - \mathbf{F}_{\text{1st}}) \cdot \mathbf{F}^{-1}_{\text{1st}}\right| \tag{16}$$

  Other types of plots involve similar dimensionless quantities.

  As an example, the following RunFile could be used as input to OVAL for creating two G-Files.

```
     0.          : rfree3   | unused in
     0.250       : dt       | time incr

       **********  Controlling Strain        krotat            iplot
        (100000)   (10000)   (1000)            |                 |
icontr| rate_11 | rate_22 | rate_33 |      igoal   finalv         |
------|---------|---------|---------|      |--|-|---------|     |--|
000000      0.        0.        0.          70 0     2.5          0
100000      0.     -0.2e-5      0.    ...   70 0  5000.0    ...   0
100000      0.     -0.2e-5      0.          70 0     5.00         1
100000      0.     -0.2e-5      0.          70 0     0.25         1
```

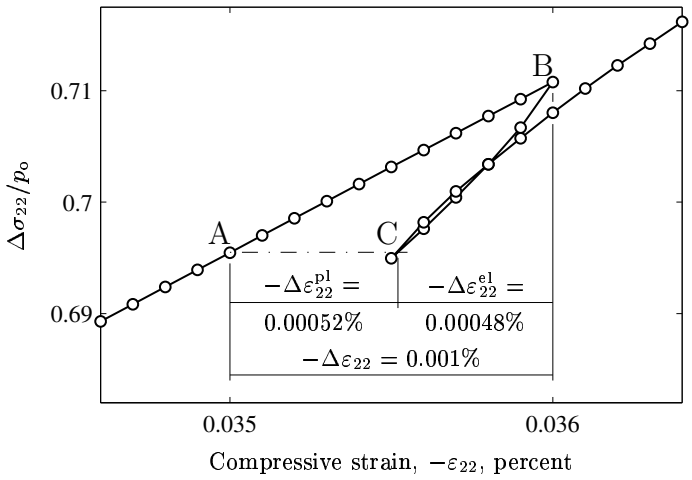


Figure 12: A cycle of loading, unloading, and reloading to produce three G-files.

The first file is created at a vertical strain of 1%, and the second file is created at a strain of 1.001%. Note that 20 time steps will separate the two G-files (= 5.00/0.250).

- Three G-files are required when OvalPlot must separate the elastic and inelastic effects. The scheme is shown in Fig. 12, which gives a detailed view of stress and strain during a cycle of loading, unloading, and reloading. The three G-files would be produced by Oval at states A, B, and C. When unloading, Oval should collect the data at a state C at which the stress is roughly the same as at state A. This may require some guessing and retrying to determine the correct number of time steps between the three states. The following run RunFile could be used to create a small cycle of loading, unloading, and reloading:

```
     0.          : rfree3   | unused in
     5.000       : dt       | time incr

       **********  Controlling Strain        krotat            iplot
        (100000)   (10000)   (1000)            |                 |
icontr| rate_11 | rate_22 | rate_33 |      igoal   finalv        |
------|---------|---------|---------|      |--|-|---------|     |--|
000000      0.        0.        0.          70 0     2.5          0
100000      0.     -0.2e-5      0.    ...   70 0   175.0    ...   0
100000      0.     -0.2e-5      0.          70 0     5.0          1
101000      0.      0.2e-5      0.          70 0     2.5          1
100000      0.     -0.2e-5      0.          70 0   100.0          1
```

In this example, the second deformation-stress segment vertically com-

presses the assembly to a strain of 0.035%. A G-file is then created at the beginning of the next (third) deformation-stress segment, and this G-file will represent state A in Fig. 12. During the third deformation-stress segment, the vertical compression continues for 10 more time steps until the vertical strain is 0.036%. Another G-file is created at the beginning of the next (fourth) deformation-stress segment, and this second G-file will represent state B in Fig. 12. The fourth deformation-stress segment unloads the assembly during 5 time steps, reducing the vertical strain to 0.0355% (note that the horizontal stress σ_{11} and shear stress σ_{12} are held constant during this unloading segment). At the beginning of the final deformation-stress segment, a third G-file is created (state C), and then the compressive loading resumes for another 20 time steps. OvalPlot calculates the proportion of elastic deformation for this cycle with the following formula:

$$\text{Proportion of elastic deformation} = \frac{\sigma^B - \sigma^A}{\sigma^B - \sigma^C} \tag{17}$$

Because of the difficulty in returning all stress components to their original values during a load cycle, the stress component specified by `istran` is used in equation 17 (see Section 16.13).

18 Running OvalPlot

Before running OvalPlot, the program will need to be installed along with a number of associated utilities (Sections 4 and 5). You will also need to create the directory (folder) into which the graphics output files will be placed, consistent with the input value of `path` in the ConfigFile (Section 16.2). You would normally run OvalPlot from within a terminal (e.g. an xterm window or DOS console) with the following command at the shell prompt:

```
<path>ovalplot
```

where `<path>` is the path to your executable `ovalplot` file. Of course, the `<path>` is not required if it is included in your system's path search definition `$PATH`.

You will then be immediately prompted for the name of your ConfigFile:

```
Name of the ConfigFile:
```

and the prefix name of the output files:

```
Name of the output file prefix:
```

You will then be queried for the type of plot that you would like to produce:

```
Which type of plot (or data) do you want to produce ?
 ** Status plots:
      1) Particle locations
      2) Particle graph of void cells
      3) Contact forces (force chains)
      4) Contact force contributions to the average stress
 ** Rate plots:
     10) Particle movements
     11) Particle rotations
     12) Particle movements and rotations
     13) Deformations within the void cells
     15) Relative particle movements at the contacts
     16) Contribution of inter-particle movements to deformation
     17) Contact force rates
     18) Contact force rate contributions to the average stress rate
     19) Particle rotations around the void cells
     20) Particle rotation gradients around the void cells
     21) Contact dislocation lines (Murakami et al.)
     22) Frictional contact sliding
     23) Energy dissipation at sliding contacts
```

The various plot types are described in Section 20. Depending upon the type of plot and whether you wish to separate the elastic/inelastic deformations, you will be asked for the names of either one, two, or three G-files (Section 17):

```
Name of the first G-type input file:
Name of the second G-type input file:
Name of the third G-type input file:
```

Depending on the type of plot, you will also be asked to select among various options (Section 21). OvalPlot will run and create a Latex `*.tex` file for producing the graphics results. You will then need to perform the remaining steps that were outlined in Section 15 for producing the screen display or printed output (items I–N, Section 15 and Fig. 10, page 66).

19 Other items on the graphics page

Besides the main graphics plot, several other items can also be placed on the graphics page. These items are optional, and their inclusion depends upon the configuration input in your ConfigFile (Section 16).

19.1 Headers and footers

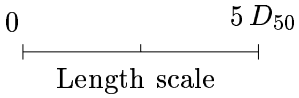
When included, the left header gives a general description of the type of file, for example,

`Mag. of right slip deformation (due to tangential movements) (rel. to avg. def.)`

The right header gives the strain at which the graphics information was collected (see `istran`, Section 16.13). The left footer lists the names of the graphics G-files that were used as input in creating the plot. The right footer includes the current date and your versions of both OVAL and OVALPLOT. Headers and footers are only included when the configuration option `iheadr=1` is given in the ConfigFile (Section 16.5).

19.2 Length scale

A length scale bar can be placed above the graphical plot. The scale bar will look something like this:

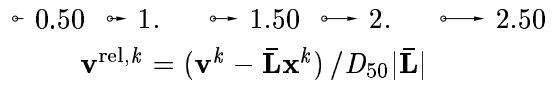


The size D_{50} is the median particle diameter, as was described in Section 12, page 59. The length scale is included only when the configuration option `iscal1=1` is given in the ConfigFile (Section 16.6).

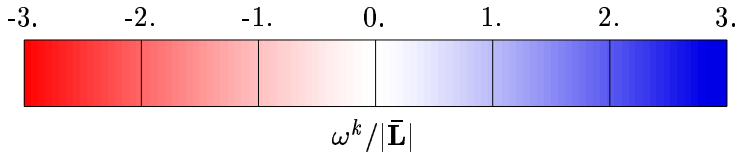
19.3 Intensity scales

OVALPLOT can produce a scale that shows the intensity of movements, stress, force, rotation, etc. Several of these are shown below.

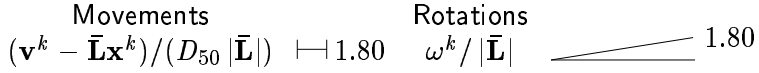
- Velocity vectors



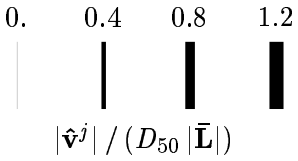
- Rotations and deformations. The width of this bar can be controlled by the input value of `sclwid` in the ConfigFile (Section 16.20).



- Velocities together with rotations



- Contact (inter-particle) movements and contact forces



The intensity scale is included only when the configuration option `iscal2=1` is given in the ConfigFile (Section 16.7).

20 Plot types in OvalPlot

Table 7 lists the various types of plots that can currently be produced with OvalPlot. You would select the type of plot at the start of an OvalPlot run. (see Section 18, page 79). The table also gives the options that are available with each plot type (refer to Section 21 for more details on each option).

The Latex output for each plot type is given a different name, of the form:

```
<your prefix><plot type name>.tex
```

where `<your prefix>` is the prefix name that you would provide as input at the start of OvalPlot (Section 18, page 78), and the `<plot type name>` is among those listed in the final column of Table 7. The ConfigFile, which was discussed in Section 16, controls the layout and style of the plots. The various types of plots are discussed below.

20.1 Particle location plots (`ktype=1`)

The assembly and its particles are drawn to scale as shown in the example plot of. Fig. 16a (page 96). You can adjust the size of the plot, provide a length scale bar, etc. by selecting the appropriate options in your ConfigFile (Section 16).

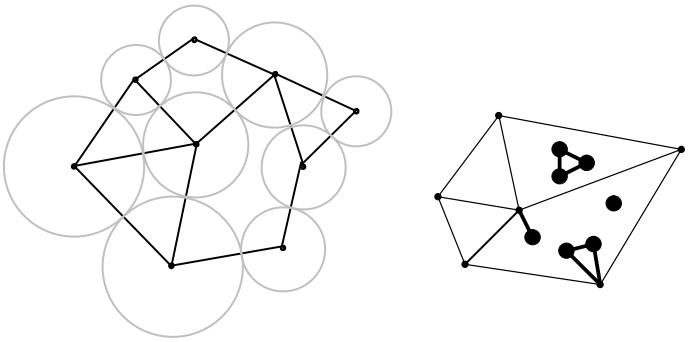
20.2 Particle graph plots of the void cells (`ktype=2`)

The assembly domain can be partitioned into numerous polygonal subdomains or *void cells*. A schematic example of the resulting *particle graph* is shown in Fig. 13a (see Satake 1992). The corners (vertices) of each polygon are the centers of particles, and the sides (edges) are the branch vectors between particle centers. The resulting particle graph includes only those particles that are in contact with neighboring particles and that participate

ktype	ilabel	idef	inorm	ifiltr	ialin	ielast	File name
1	√	NA	NA	NA	NA	NA	*_part_posn.tex
2	√	NA	NA	NA	NA	NA	*_graph.tex
3	–	NA	◯	NA	NA	NA	*_contact_forc.tex
4	–	NA	◯	√	√	NA	*_contact_stress.tex
10	–	√	NA	NA	NA	√	*_part_move.tex
11	√	√	NA	NA	NA	√	*_part_rotat.tex
12	–	√	NA	NA	NA	√	*_move_rotat.tex
13	√	√	◯	√	√	√	*_cell_def.tex
15	–	√	◯	NA	NA	√	*_contact_move.tex
16	–	√	√	√	√	–	*_contact_def.tex
17	–	√	◯	NA	NA	–	*_contact_dforc.tex
18	–	–	◯	√	√	–	*_contact_dstres.tex
22	–	–	◯	NA	NA	NA	*_contact_slip.tex
23	–	–	◯	NA	NA	NA	*_contact_dis.tex

ktype	Plot type
ilabel	Labeling of particles and void cells, Sections 16.10 and 21.1
idef	Plotting either the absolute movements **or** the movements relative to the assembly deformation, Section 21.2
inorm	Plotting of inter-particle movements either tangent or normal to the contact surface, Section 21.3
ifiltr	Applying filters to deformations and stresses, Section 21.4
ialin	Plotting of either a filtered magnitude or an alignment relative to the filter, Section 21.5
ielast	Plotting of the separate elastic and inelastic movements/deformations, Sections 16.15, 17, and 21.6
NA	Not applicable
◯	For circular particles only, not ellipses or ovals

Table 7: Summary of plot types and options



(a) Particle graph (b) Non-participating particles (disregarded)

Figure 13: Particle graph of a 2D assembly

in the load-bearing framework of the assembly. Island, peninsula, and pendant particles are ignored by OvalPlot (Fig. 13b). Figure 16b (page 96) shows then entire particle graph of an assembly of 1002 particles.

20.3 Contact forces (`ktype=3`)

The magnitudes of the contact forces are represented by adjusting the thicknesses of lines in the particle graph (Fig. 18a, page 98). OvalPlot represents the contact force $\mathbf{f}^k$ at the k^{th} contact in the following dimensionless form:

$$|\mathbf{f}^k|/(p\,D_{50}) \tag{18}$$

where D_{50} is the median particle diameter and p is the current mean stress. The program gives the option of plotting the contributions of either the total contact force or of the tangential or normal components alone (Section 21.3). Except when plotting tangential force components, the forces are represented with black lines. When plotting tangential components, a color scale is used, in which blue represents a positive tangential force and red represents a negative tangential force (as in Fig. 18a, page 98). The sign convention for tangential forces is illustrated in Fig. 14: a positive tangential force will tend to spin each of the two particles in a counterclockwise direction.

20.4 Contact force contributions to the average stress (`ktype=4`)

If $\mathbf{f}^k$ and $\mathbf{l}^k$ are the contact force and branch vector for the k^{th} contact, then the magnitude of its contribution to the average assembly stress $\overline{\boldsymbol{\sigma}}$ is the inner product

$$(\mathbf{f}^k \otimes \mathbf{l}^k) : \overline{\boldsymbol{\sigma}}/(D_{50}^2|\overline{\boldsymbol{\sigma}}|) \tag{19}$$

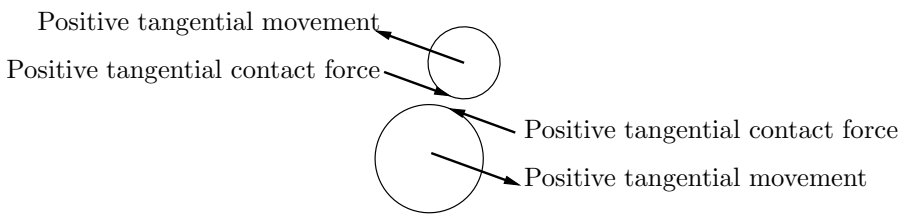


Figure 14: Sign convention for tangential contact forces and movements. Positive values are plotted with blue lines; negative values are plotted with red lines (see Fig. 18a, page 98).

where $\overline{\boldsymbol{\sigma}}$ is the current average stress tensor for the entire assembly, and the tensor norm is defined as $|\overline{\boldsymbol{\sigma}}| = (\overline{\sigma}_{ij}\overline{\sigma}_{ij})^{1/2}$. OvalPlot gives the option of plotting the *magnitude* of the contribution in eqn. (19) or the "alignment" of the contribution $\mathbf{f}^k \otimes \mathbf{l}^k$ with the stress $\overline{\boldsymbol{\sigma}}$ (Section 21.5). The alignment will range from -1 to $+1$, and it is computed as

$$(\mathbf{f}^k \otimes \mathbf{l}^k) : \overline{\boldsymbol{\sigma}} / (|\mathbf{f}^k \otimes \mathbf{l}^k|\,|\overline{\boldsymbol{\sigma}}|) \ . \tag{20}$$

When plotting alignments, a red/blue color scale is used to represent negative/positive values over the range -1 to $+1$. OvalPlot also allows you to plot the contribution of the contact to either the current stress $\overline{\boldsymbol{\sigma}}$ or to a "filter" stress $\boldsymbol{\Phi}$ (e.g. mean stress, deviatoric stress, etc., see Section 21.4).

20.5 Particle movements (`ktype=10`)

The particle movements are plotted with arrow vectors (Fig. 17a, page 97). Each arrow vector is scaled to the magnitude of a particle's movement, and the arrow lengths will usually be greatly magnified (see Section 16.17 to adjust the scaling). The arrows represent the difference in particle locations between two G-files (Section 17), which are presented as the dimensionless rate

$$\mathbf{v}^k / (D_{50} |\overline{\mathbf{D}}|) \ , \tag{21}$$

where $\overline{\mathbf{D}}$ is the assembly's average rate of deformation (the symmetric part of the velocity gradient, $\overline{\mathbf{L}}$) as computed from the change in the deformation gradient: $\overline{\mathbf{L}} = \dot{\mathbf{F}} \cdot \mathbf{F}^{-1}$. The tensor norm is defined as $|\overline{\mathbf{D}}| = (\overline{\mathbf{D}}{:}\overline{\mathbf{D}})^{1/2}$. Particles without contacts are not plotted (see Fig. 13b). OvalPlot gives the option of plotting the actual movements or the movements relative to the mean-field deformation (Section 21.2, as has been proposed by Williams and Nabha 1997). You can also separately plot the elastic and inelastic movements (Sections 16.15, 17, and 21.6).

20.6 Particle rotations (`ktype=11`)

The particle rotations are plotted by shading each particle in proportion to its rotation magnitude (Fig. 18b, page 98)

$$\omega^k/|\overline{\mathbf{D}}| \ . \tag{22}$$

Clockwise rotations are shaded red; counterclockwise rotations are shaded blue. Particles without contacts are not displayed at all (see Fig. 13b). OvalPlot gives the option of plotting the actual rotations or the rotations relative to the mean-field rotation (Section 21.2). You can also separately plot the elastic and inelastic rotations (Sections 16.15, 17, and 21.6). You also have the option of plotting both positive and negative rotations, only positive rotations, or only negative rotations (the latter two cases are useful if the figures are to appear in a monochrome format). These various options are presented on the screen while you are running OvalPlot.

20.7 Combined particle movements and rotations (`ktype=12`)

Both movements and rotations are displayed by showing the particles in their initial and final positions (shaded and solid, respectively) and with the initial and displaced branch vectors of all contacts (Fig. 17b, page 97). These plots can be a bit dense, so you might want to experiment with plotting a subset of the entire assembly (Section 16.16).

20.8 Void cell deformations (`ktype=13`)

OvalPlot can compute and plot the deformations within the polygonal void cells of an assembly (see Section 20.2 for a description of the particle graph and void cells). These polygonal regions are deformed as the particle centers move, and the methods for computing these deformations have been presented by Bagi (1996) and Kuhn (1999). OvalPlot uses two G-files to compute the average velocity gradient $\overline{\mathbf{L}}^i$ within each (i^{th}) void cell. Because it is impossible to display all four components of this tensor, you will instead display a single value that has been "filtered" with the inner product

$$\overline{\mathbf{L}}^i{:}\mathbf{\Phi} \ , \tag{23}$$

which produces the length of $\overline{\mathbf{L}}^i$ in the direction of the filter tensor $\mathbf{\Phi}$. Various filters are offered as options (dilation, simple shear, inclined slip, etc., see Section 21.4). OvalPlot displays the dimensionless rate

$$\overline{\mathbf{L}}^i{:}\mathbf{\Phi}/(|\overline{\mathbf{D}}^i||\mathbf{\Phi}|) \ , \tag{24}$$

where $\mathbf{L}$ is the velocity gradient, $\mathbf{D}$ is the rate of deformation tensor $|\overline{\mathbf{D}}|$ and $|\mathbf{\Phi}|$ are tensor norms, and $\overline{\mathbf{D}}$ is the mean-field deformation of the entire assembly. A tensor norm is defined as

$$\mathbf{P} = \sqrt{P_{ij}P_{ij}} \tag{25}$$

Figure 18c on page 98 shows the right slip deformations that have occurred in an assembly of 1002 particles. (See Fig. 15, page 89, for illustrations of left slip and right slip deformations.)

You can plot the separate effects of the normal and tangential inter-particle movements on the void cell deformations (Section 21.3). OvalPlot also gives the option of plotting either the actual deformations or the deformations relative to the mean-field deformation of the entire assembly (Section 21.2). You can separately plot the elastic and inelastic deformations (Sections 16.15, 17, and 21.6). You also have the option of plotting both positive and negative values, only positive values, or only negative values (the latter two cases are useful if the figures are to appear in a monochrome format). These various options are presented on the screen while you are running OvalPlot.

20.9 Inter-particle movements at contacts (`ktype=15`)

OvalPlot can depict the relative movements that occur between pairs of contacting particles. These relative, inter-particle movements include the relative translations of the two particles, but not their rotations (future versions of OvalPlot will include the ability to depict rotation effects upon the contacts).

You can plot the separate effects of the normal and tangential inter-particle movements on the void cell deformations (Section 21.3). When plotting the normal inter-particle movements, positive (blue) movements tend to reduce the normal contact force. When plotting the tangential inter-particle movements, positive and negative movements are as shown in Fig. 14. OvalPlot also gives the option of plotting the actual inter-particle movements or the movements relative to the mean-field deformation of the entire assembly (Section 21.2). You can also separately plot the elastic and inelastic inter-particle movements (Sections 16.15, 17, and 21.6).

20.10 Contributions of inter-particle movements to the average deformation (`ktype=16`)

With this plot type, OvalPlot computes an approximate measure of the contribution that each inter-particle movement makes to the deformation of the assembly. The method is based on the observation that each contact is the side (edge) of two polygonal void cells (see Section 20.2 and Fig. 13,

page 83). We can compute the deformation that is produced in each of the two void cells by the single inter-particle movement of the two contacting particles. This computation uses the methods of Bagi (1996) and Kuhn (1999), but only a single contact movement is used in the calculation of a void cell's deformation. The method provides only a rough estimate of the contribution of an inter-particle (contact) movement to average deformation of the assembly. You can apply the same options to these plots as those discussed in Section 20.8 (`ktype=13`).

20.11 Contact force rates (`ktype=17`)

The changes in the contact forces are plotted in a manner similar to that of plotting the (static) contact forces (`ktype=3`, Section 20.3). The changes in the forces are computed by comparing two G-files (Section 17). The changes are presented as the dimensionless rates

$$\dot{\mathbf{f}}^k/(p\,D_{50}\,|\overline{\mathbf{D}}|)\;, \tag{26}$$

which is similar to eqn. (18) except for the use of the divisor $|\overline{\mathbf{D}}|$. The options that were discussed in Section 20.3 also apply to plots of the contact force rates.

20.12 Contact force rates contributions to the average stress rate (`ktype=18`)

OvalPlot can plot the contributions that changes in the contact forces make to the change in the average assembly stress. This type of plot is similar to the one described in Section 20.4, except that force rates, instead of forces are used in the calculations. The changes in force are computed from two G-files. The dimensionless rate contributions are computed as

$$(\dot{\mathbf{f}}^k \otimes \mathbf{l}^k) : \dot{\overline{\boldsymbol{\sigma}}}/(D_{50}^2|\dot{\overline{\boldsymbol{\sigma}}}|) \tag{27}$$

Note that the rate in eqn. (27) is only approximate, since the rates $\dot{\mathbf{l}}^k$ are not included.

The options that were discussed in Section 20.4 also apply to this plot type.

20.13 Frictional contact sliding (`ktype=22`)

This plot type identifies those contacts at which frictional sliding occurs. Only the branch vectors of sliding contacts are plotted. The thickness of a branch vector is proportional to the rate of frictional sliding. Red and blue

lines correspond to negative and positive sliding (see Fig. 14). Sliding rates are presented in a dimensionless form as

$$(\text{sliding rate})/(D_{50}|\overline{\mathbf{D}}|) \; . \tag{28}$$

20.14 Energy dissipation at sliding contacts (`ktype=23`)

Sliding contacts are identified with solid black lines whose thicknesses are proportional to the rate of energy dissipation at these contacts. The following dimensionless rate is used:

$$(\text{energy dissipation rate})/(p\, D_{50}^2|\overline{\mathbf{D}}|) \; . \tag{29}$$

21 OvalPlot options

OvalPlot offers several options for analyzing and plotting various micro-quantities. Table 7 (page 82) summarizes the availability of these options for different plot types and particles shapes. Most of these options are presented at the command prompt when you run OvalPlot. The ConfigFile also provides a number of options that control the layout of a plot (Section 16). The options in Table 7 are discussed below.

21.1 Labels on particles and void cells (`ilabel`)

Number labels can be placed on particles and void cells. See Section 16.10.

21.2 Actual or relative movements (`idef`)

You can plot either the actual movements and deformations (`idef=0`) or the movements and deformations that are relative to the mean-field deformation of the entire assembly (`idef=1`). The option is presented as follows:

```
Which movements/deformations will be used (idef)?
  0) Actual movements
  1) Deformation relative to the global (mean-field) movement
```

21.3 Total, normal, or tangential inter-particle effects (`inorm`)

You will be prompted with options such as the following:

```
Which components of inter-particle movements are included (inorm)?
  0) total
  1) normal
  2) tangential
```

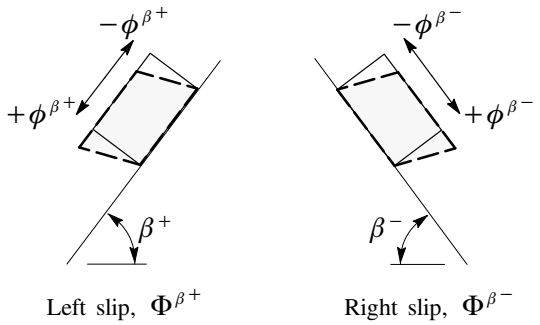


Figure 15: Left and right slip deformation modes.

The normal and tangential directions are referenced to the contact surfaces. The directions of positive inter-particle movements and forces are shown in Fig. 14.

21.4 Filters (`ifiltr`)

For movements and deformations, you can choose among the following filters ($\mathbf{\Phi}$, see Section 20.8):

```
Which filter will be used (ifiltr)?
  1) the mean-field (global) deformation
  2) rotation [0 -1;1 0]
  3) dilation [1  0;0 1]
  4) right slip deformation
  5) left slip deformation
  6) shear [0 1;1 0]
  7) your filter (to be given as input)
  8) simple shear [0 1;0 0]
  9) simple shear [0 0;1 0]
```

If you choose right slip or left slip deformations (items 4 and 5), you will be prompted for the slip angle β (see Fig. 15). If you choose item 7, "your filter," you will be prompted for the components of $\mathbf{\Phi}$ in the following order: ϕ_{11}, ϕ_{12}, ϕ_{21}, ϕ_{22}. Enter all four values on a single line, with the items separated with commas or spaces.

With forces or stresses, you can choose among the following filters ($\mathbf{\Phi}$, see Section 20.4):

```
Which filter will be used (ifiltr)?
 1) the average stress stress within the assembly
 2) un-symmetric stress [0 1;-1 0]
 3) mean stress        [1 0; 0 1]
```

```
4) right slip shearing
5) left slip shearing
6) pure shear stress  [0 1; 1 0]
7) your filter (to be given as input)
```

Items 4 and 5 refer to shearing stresses in the manner shown in Fig. 15.

21.5 Magnitudes or alignments (ialin)

When a filter $\mathbf{\Phi}$ is being applied to a tensor quantity, say $\mathbf{A}$, you can either plot the inner product magnitude $\mathbf{A} : \mathbf{\Phi} = A_{ij}\Phi_{ij}$ or the alignment of the two tensors, $\mathbf{A} : \mathbf{\Phi}/(|\mathbf{A}||\mathbf{\Phi}|)$. You will be offered a choice with the following prompt:

```
Magnitude  or  alignment (ialin)?
 0) magnitude, A:B
 1) alignment, A:B/(|A| |B|)
```

21.6 Elastic and inelastic movements (ielast)

This option was discussed in Sections 16.15 and 17. The option is given in the ConfigFile, and it requires three G-files as input. If the ConfigFile includes the input value `iplast=1`, then you will be prompted with the following choices:

```
Which of the following do you want to plot (ielast)?
  1) elastic
  2) inelastic
  3) total
```

22 Change Log

This section documents the changes that have been made between various version of Oval and OvalPlot.

22.1 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 8.1.25).
- Added the option of `nloop1` into the input RunFile (Section 8.1.13).

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".

22.2 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 3)
- 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.

22.3 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.

22.4 Oval–0.6.2 to Oval–0.6.3

Bug fixes:

- Changes to contact inquiry algorithm for ovoids.

22.5 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 accomodate larger simulations.

Bug fixes:

- Slight change in algorithm for contact damping.

22.6 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.

22.7 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- nad 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`.

22.8 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.

22.9 OvalPlot–0.2.0 to OvalPlot–0.4.0

Added features:

- Added the option of printing only positive or negative values when `ktype=11` or `ktype=13`.
- Added the configuration value `sclwid` to the ConfigFile, which allows adjusting the width of the intensity scale.

Bug fixes:

- Changed the type of parameter "mfirst" to integer*4.
- Changed plots of contact sliding (`ktype=22`) so that non-sliding contacts are not plotted (they previously appeared as faint pink lines).

22.10 OvalPlot–0.4.0 to OvalPlot–0.4.2

Enhancements:

- Change from **L** to **D** to measure certain rates.

23 References

Bagi, K. (1996). "Stress and strain in granular assemblies." *Mech. of Materials*, 22(3), 165–177.

Bardet, J. P. (1994). "Observations on the effects of particle rotations on the failure of idealized granular materials." *Mech. of Materials*, 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.

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

Murakami, A., Sakaguchi, H., and Hasegawa, T. (1997). "Dislocation, vortex and couple stress in the formation of shear bands under trapdoor problems." *Soils and Found.*, Jap. Geotech. Soc., 37(1), 123–135.

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

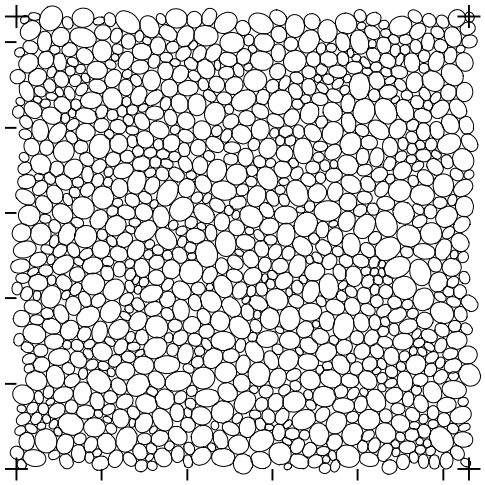
Satake, M. (1982). "Fabric tensor in granular materials." *Proc. IUTAM Symp. on Deformation and Failure of Granular Materials*, P. A. Vermeer and H. J. Luger, eds., A.A. Balkema, Rotterdam, 63–68.

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

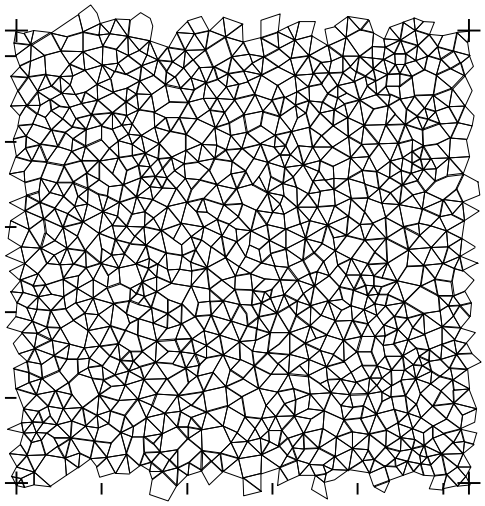
Satake, M. (1993). "New formulation of graph-theoretical approach in the mechanics of granular materials." *Mech. of Materials*, 16, 65–72.

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.

Williams, J. R. and Nabha, R. (1997). "Coherent vortex structures in deforming granular materials." *Mech. Cohesive-Frictional Mat.*, 2(3), 223–236.

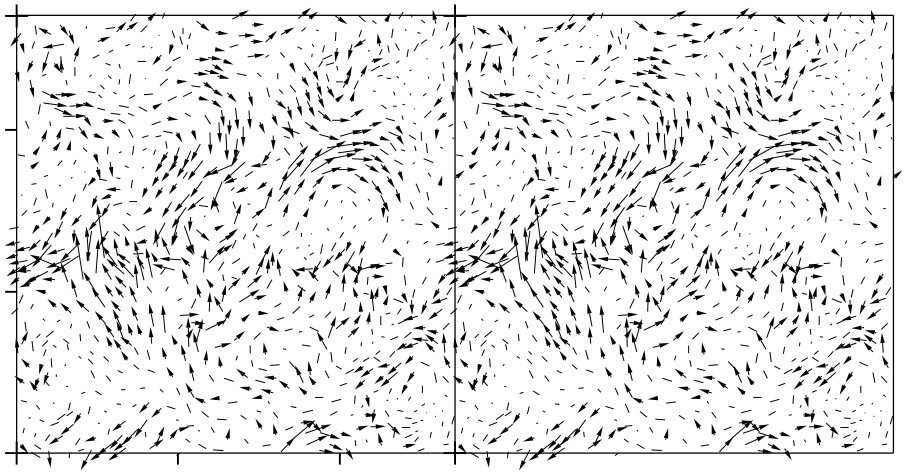


(a) Particle positions, oval particles (`ktype=1`, Section 20.1).

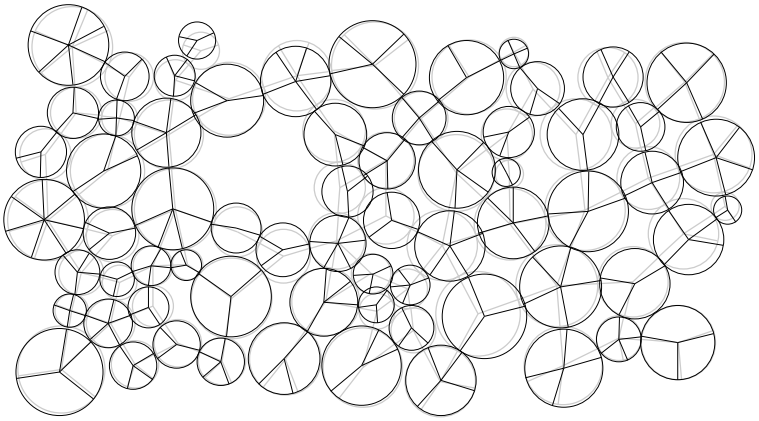


(b) Particle graph (`ktype=2`, Section 20.2).

Figure 16: Example plots with an assembly of 1002 particles.

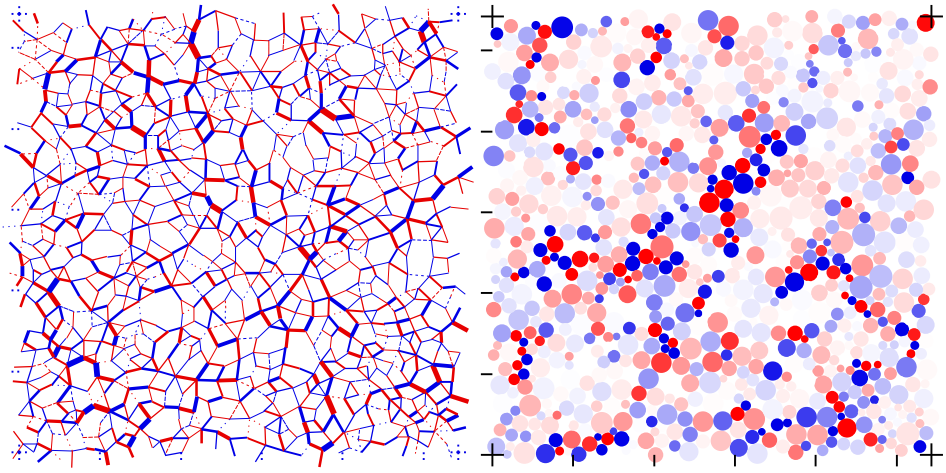


(a) Particle movements (`ktype=10`, Section 20.5). Two assemblies are tiled side-by-side.



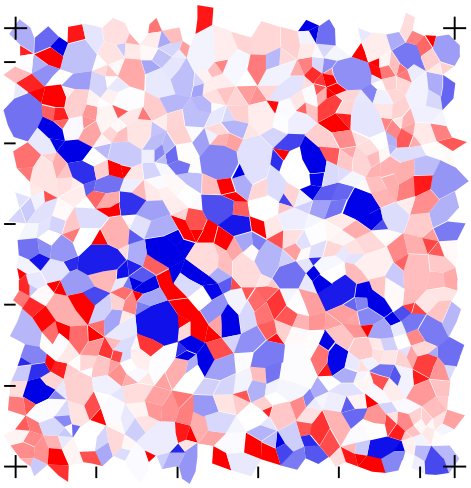
(b) Particle movements and rotations (`ktype=12`, Section 20.7). A subset of the entire assembly is shown.

Figure 17: More example plots with an assembly of 1002 particles.



(a) Tangential contact forces (`ktype=3`).

(b) Particle rotations (`ktype=11`, Section 20.6).



(c) Right slip deformations (`ktype=13`, Section 20.8).

Figure 18: More example plots with an assembly of 1002 particles.