Dynamo

Command line functionalities: express tutorial

Command line tools

* Sometimes they are the only option:
  - GUIs can become too slow in remote systems.
  - GUIs cannot cover the whole flexibility of the environment.
* Clearer insight on the underlying workflows
* Requirement for own developments (plugins, scripts,...)
* Faster operation
  - ... once you are familiar with syntax and commands

After a smooth learning curve

What is the “command line”?

you have two options, which are equivalent for the purposes of this tutorial)

1- The Matlab shell
A regular Matlab shell, assuming of you course that
a) you have activated *Dynamo*)
b) you have a Matlab license (no further toolboxes are strictly required)

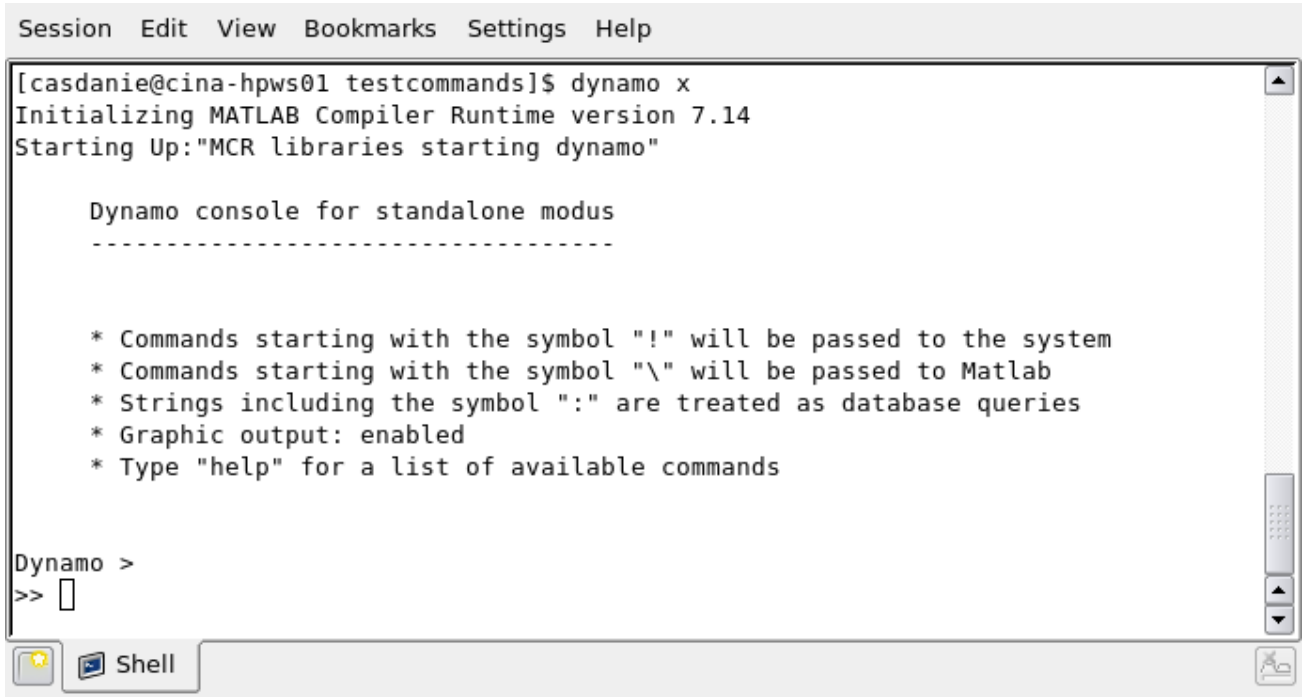
2- The *Dynamo* console
Completely independent on Matlab licenses (but needs the free MCR libraries installed)

Notes on the *Dynamo* console:

1- opens with
`dynamo x`
after you activate *Dynamo* in your system

2- Can take a while when opening (as you are waking up Matlab and *Dynamo*)

3- The first commands run from the console while also need some time to start up.



Syntax basics

Most Dynamo commands can be expressed in two syntax types:

Functional syntax:

- Typical Matlab syntax.
- Valid only in Matlab prompts

EXAMPLE (abstract example: do not type it right now!)

```
volsym = dynamo_sym(my_vol,'c1','fmask',my_fmask);
```

String syntax:

- Typical Unix-like syntax.
- Valid both in Matlab prompts and in Dynamo standalone consoles:

EXAMPLE:

```
dsym vol.em  c1 -fmask fmask.em -ws volsym;
```

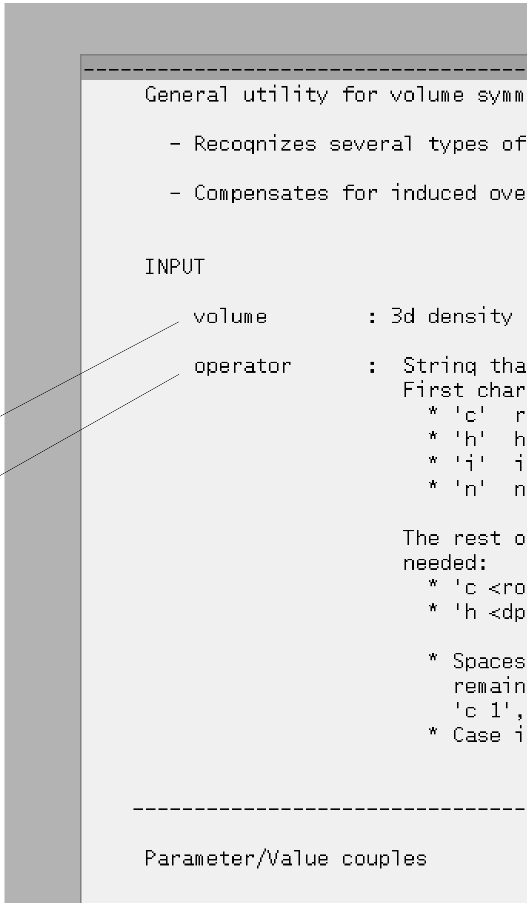
Each command has its own syntax:

get particular information with:

`help dsym` [command line help]

`doc dsym` [Matlab help browser if working in Matlab /Dynamo help browser if working in the Dynamo console]

`ddoc dsym` [Dynamo help browser]



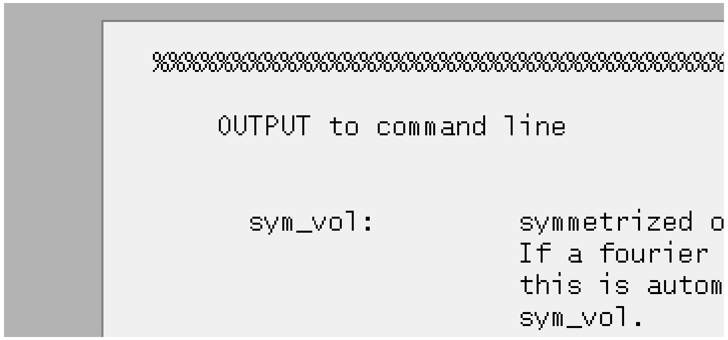
```
---------------------------------------
   General utility for volume symm

     - Recognizes several types of

     - Compensates for induced ove


   INPUT

       volume        : 3d density

       operator      :  String tha
                        First char
                          * 'c'   r
                          * 'h'   h
                          * 'i'   i
                          * 'n'   n

                        The rest o
                        needed:
                          * 'c <ro
                          * 'h <dp

                          * Spaces
                            remain
                            'c 1',
                          * Case i


  -------------------------------

   Parameter/Value couples
```

In this case, the part of the help describing the syntax tells you that the two first positions are reserved for:
* the volume and
* the symmetry operator
(in this order).

```
volsym = dynamo_sym(my_vol,'c1','fmask',my_fmask);
```

output to command line:

a variable will be created in the workspace of Matlab



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    OUTPUT to command line


      sym_vol:         symmetrized o
                       If a fourier
                       this is autom
                       sym_vol.
```

everything else in the right hand side are couples of Parameter / Value (in arbitrary order):

`'fmask'` is the name of the parameter

`my_fmask` is the value we pass to this parameter

String notation is similar:

but notice:

all the inputs are strings by default:
you don't need the quote notation (`'c1','fmask'`,etc)

```
dsym vol.em  c1 -fmask fmask.em -ws volsym;
```

Natural input objects are filenames.

You can still pass variable names (preceded by @)
but it is not really comfortable.

there isn't a left hand side any more:
the output is controlled by Parameter/Values

In this case 'ws' will create the variable 'volsym' in the workspace,
exactly as in functional notation case.

So let'us play a little with simple objects.

Create a new directory (say, 'testcommands') and go there:

```
>> mkdir testcommands;
>> cd testcommands;
```

We now explore the contents of the directory (from the Dynamo point of view)

```
>> dsummary;
```

This command looks for typical objects created during Dynamo sessions.
The output that you should get (depending on version) is pasted in the next slide:
We just started working, so it is nothing there yet.

However, this simple command tends to be very useful when visiting back your work after a couple of days.

Note:
the “`Executing`” and “`Exiting`” messages appear only in the *Dynamo* console, not when working directly on a Matlab shell.

```
Session   Edit   View   Bookmarks   Settings   Help

Dynamo >  >> pwd
/home/casdanie/test/testcommands
Dynamo >  >> dsummary
"dsummary" references source command "summary"

------------------------------------------------------------
  Executing:  summary

------------------------------------------------------------

 Gathering a summary of Dynamo contents in folder "."

projects              : 0
data folders          : 0
subboxing folders     : 0
bundles               : 0
tutorial folders      : 0

  [ok] summary

------------------------------------------------------------

  Exiting:  summary
------------------------------------------------------------

Dynamo >  >>

Shell
```

do not worry about “subboxing” and “bundles”, and let us move forward

So let us create some data to start playing:

A folder with files to define easily all elements of a subtomogram averaging experiment:

```
dtutorial test -p ptest
```

we simultaneously request (“-p”) the creation of a project arbitrarily called “ptest”, which will be fed with the files generated inside the folder “test”

... you will get a lot of on-screen information about what is happening...-

Session Edit View Bookmarks Settings Help

```
-------------------------------------------------
Dynamo was not able to set execution permissions on the execution script
You may need to do it manually with chmod u=rwx test.exe
 ...done: general project "test" ready for execution using script:
  test.exe

 CCMATRIX PROJECT

Creating a ccmatrix project "ccmatrix_test" fed with the real alignment parameters "test"

Checking virtual project: (command: "dynamo_vpr_check)

... checking passed

Unfolding virtual project: (command: "dynamo_vpr_unfold")
Empty field options
 ...done: ccmatrix project "ccmatrix_test" ready for execution using script:
  ccmatrix_test.m


  Exiting:  tutorial
-------------------------------------------------------------

Dynamo >  >>
```

Shell

... when it is done, if you run again the `summary` command:

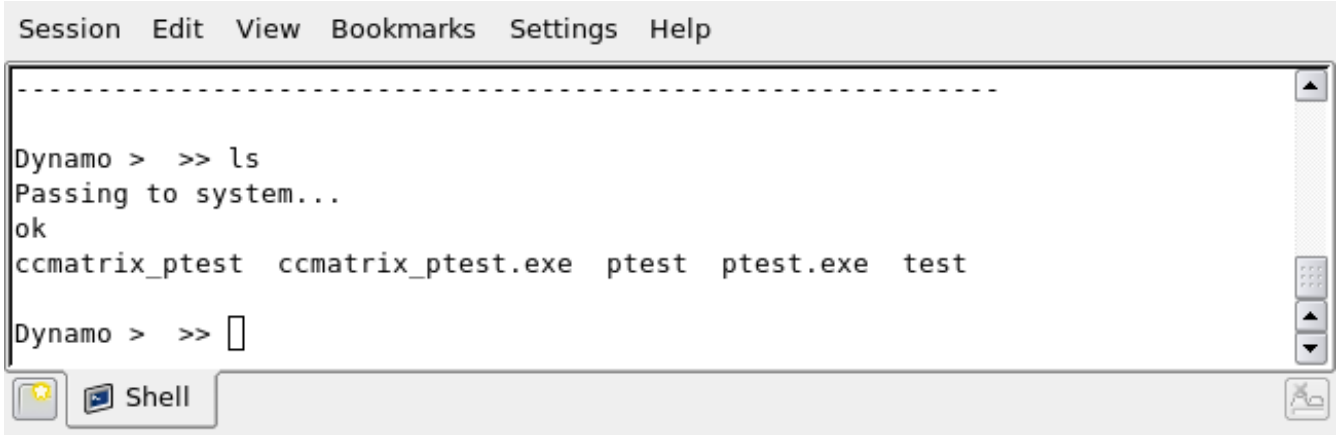
```
Dynamo >  >> dsummary
"dsummary" references source command "summary"

------------------------------------------------------------
  Executing:  summary

------------------------------------------------------------

 Gathering a summary of Dynamo contents in folder "."

projects              : 2
data folders          : 0
subboxing folders     : 0
bundles               : 0
tutorial folders      : 1

  [ok] summary

------------------------------------------------------------

  Exiting:  summary
------------------------------------------------------------

Dynamo >  >>
```

some entities appear: two “projects” and one “tutorial folder”

you can activate a deeper level of detail as second argument of `dsummary`

```
Dynamo >  >> dsummary . on
"dsummary" references source command "summary"

-------------------------------------------------------------
  Executing:  summary

Parsing 2 arguments
-------------------------------------------------------------

 Gathering a summary of Dynamo contents in folder "."

projects              : 2
                        ccmatrix_ptest
                        ptest
data folders          : 0
subboxing folders     : 0
bundles               : 0
tutorial folders      : 1
                        ./test

  [ok] summary

-------------------------------------------------------------

  Exiting:  summary
-------------------------------------------------------------

Dynamo >  >>
```

They are [normally] just folders, visible through regular ls/dir actions:



but summary categorizes everything within the Dynamo framework.

Other commands allow you to focus on specific categories:
`projects, data, bundles, subboxing, tutorials`

Type for instance:
`>> dprojects`

```
Session   Edit   View   Bookmarks   Settings   Help
-----------------------------------------------------------

Dynamo >  >> ls
Passing to system...
ok
ccmatrix_ptest  ccmatrix_ptest.exe  ptest  ptest.exe  test

Dynamo >  >> dprojects
"dprojects" references source command "projects"

-----------------------------------------------------------
  Executing:  projects

Found 2 Dynamo projects in location .:
ccmatrix_ptest   ptest

  Exiting:  projects
-----------------------------------------------------------

Dynamo >  >>
Shell
```

You get a list of available projects.

Note that dtutorial actually produced two projects:
`ptest` is a “regular” project for alignment of a data set.
`ccmatrix_ptest` is a project that targets a classification (by computing a ccmatrix)

Likewise for the tutorial objects. Type
>> tutorials

```
Session   Edit   View   Bookmarks   Settings   Help

Dynamo >  >> dtutorials
"dtutorials" references source command "tutorials"

-------------------------------------------------------------
  Executing:  tutorials

-------------------------------------------------------------
 A total of 1 tutorial folders:
folder                M    N    project
./test                8    0    'ptest'

  [ok] tutorials

-------------------------------------------------------------

  Exiting:  tutorials
-------------------------------------------------------------

Dynamo >  >>

Shell
```

the list generated by tutorials just informs on the created particles (M and N) and the companion project

“tutorials” have in fact lots of options to allow the creation of simulations that focus on different aspects of subtomogram averaging.

doc tutorial

will present a list of possibilities as:

Particle sizes
Templates
Fourier sampling
Noise
Geometric constraints.

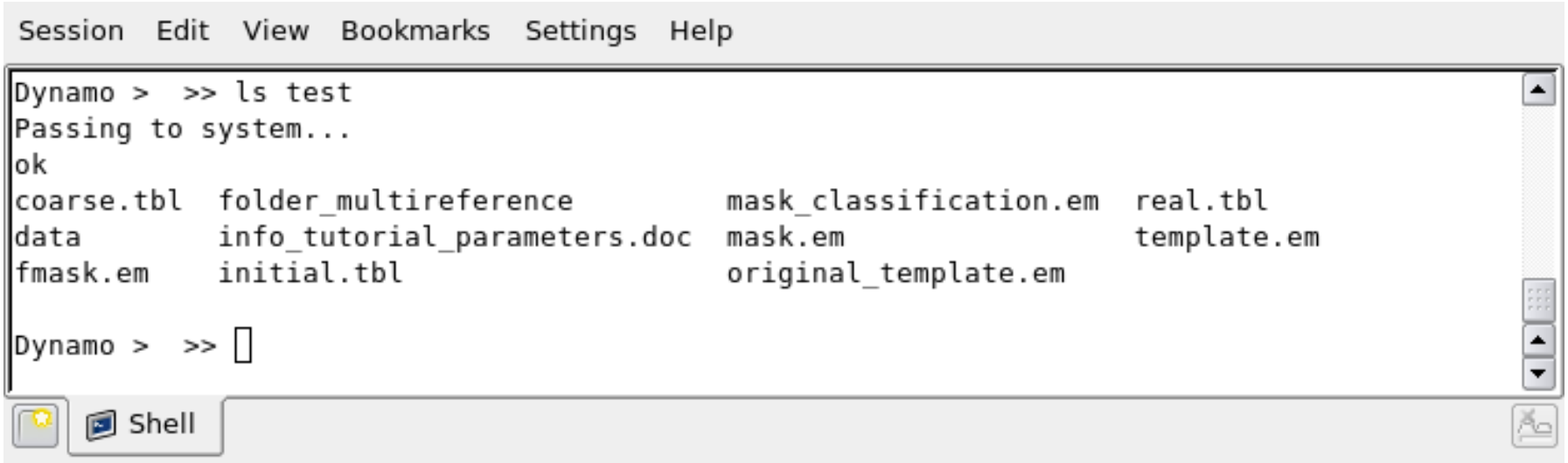
You can always recall the creation settings of a present tutorial by looking at its “info” file.

Session Edit View Bookmarks Settings Help

```
Dynamo >  >> type test/info_tutorial_parameters.doc

               bin: 0
              cone: 0
       data_output: 1
       destination: 'system_omp'
            dshift: 3
             dtilt: 3
       empty_space: ''
         extension: '.em'
            folder: 'test'
             ftype: 'single'
                jd: []
                js: []
          jsubtomo: 0
         just_data: 0
        just_shift: 0
          location: ''
                 M: 8
    multireference: []
                 N: 0
             noise: 0.1000
                 p: 'ptest'
           project: 'ptest'
       real_random: 0
          template: []
         template1: 'thermo1.em'
         template2: 'thermo2.em'
             tight: 0
              tilt: 60
                ws: []
```

Ok, then... what is in our tutorial `"test"`?

```
>> ls test
```



we see several "table" files, (with extension .tbl)

`real.tbl`
describes the real geometric configuration of the synthetic particles.
the results of an alignment project should approach this table.

`initial.tbl`
is just a blank table that covers the synthetic particles. It is intended to be uses as seed for an alignment project

`coarse.tbl`
a perturbation of `real.tbl`

TABLES

dynamo_table_info
dtinfo

it is a practical way to get at once
an idea of what is inside a table file:

Session Edit View Bookmarks Settings Help

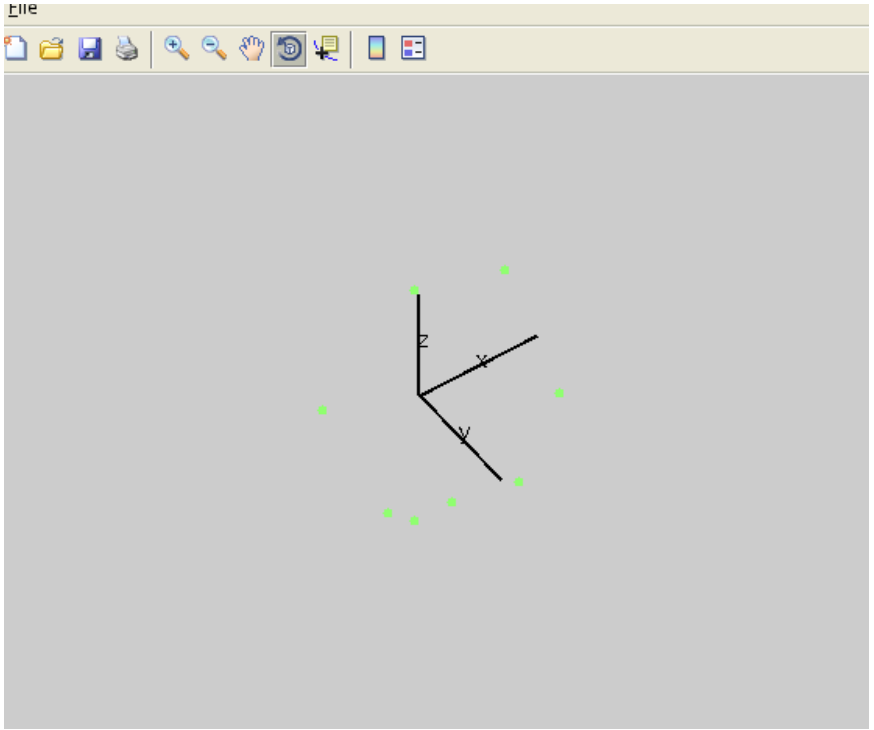
```
Dynamo >  >> dtinfo test/real.tbl
"dtinfo" references source command "table_info"

-----------------------------------------------------------
  Executing:  table_info


 table file: test/real.tbl
        size                :  8 26
        NaNs                :  0

 COLUMN
[  2  ] marked for alignment:  8
[  3  ] included in average :  8
[ 4-6 ] shifts              :  initialized: 8
[  4  ]   *  x              :  min: -1.58  max: 7.49  mean: 1.68  std: 2.92
[  5  ]   *  y              :  min: -6.29  max: 1.48  mean: -2.32  std: 2.84
[  6  ]   *  z              :  min: -2.45  max: 4.88  mean: 0.76  std: 2.90
[ 7-9 ] angles              :  initialized: 8
[  7  ]   *  tdrot          :  min: -104.50  max: 156.90  mean: 27.22  std: 90.06
[  8  ]   *  tilt           :  min: -167.54  max: 140.60  mean: -22.78  std: 116.42
[  9  ]   *  narot          :  min: 12.31  max: 101.90  mean: 70.38  std: 29.59
[  10 ] cross correlation   :  min: 0.00  max: 0.00  mean: 0.00  std: 0.00
[  13 ] Fourier sampling    :  1 (single tilt around y)
[  13 ] fsampling types     :  all of the same type
[14-15] ytilt range         :  min:120.00   max:120.00
[16-17] xtilt range         :  min:120.00   max:120.00
[  20 ] linked volumes      :  total 1 (labels: [0])
[  22 ] user-defined classes:  total 1 (labels: [0])
[  23 ] annotation types    :  total 1 (labels: [0])
[24-26] spatial locations   :  initialized: 8
[  24 ]   *  x              :  min: 58.07  max: 246.62  mean: 153.07  std: 73.82
[  25 ]   *  y              :  min: 33.97  max: 265.07  mean: 138.92  std: 74.40
[  26 ]   *  z              :  min: 22.80  max: 294.81  mean: 137.24  std: 115.46
[  27 ] diff. shift         :  Warning: column not available in this table
[  28 ] diff. axis          :  Warning: column not available in this table
[  29 ] diff. narot         :  Warning: column not available in this table
[  34 ] references          :  Warning: column not available in this table
[  35 ] subreferences       :  Warning: column not available in this table
[  36 ] apix                :  Warning: column not available in this table
[  37 ] defocus             :  Warning: column not available in this table

  Exiting:  table_info
-----------------------------------------------------------

Dynamo >  >>
```

Shell

But we can also focus on some aspect of the table for a more detailed depiction.

A basic command to view the orientations of your particles is:

```
>> dtplot test/real.tbl
```



in this depiction modus of `dynamo_tableplot` (or `dtplot`) each point in the unit sphere represents the direction of a particle as determined in the table (columns 7:9 store the angles).

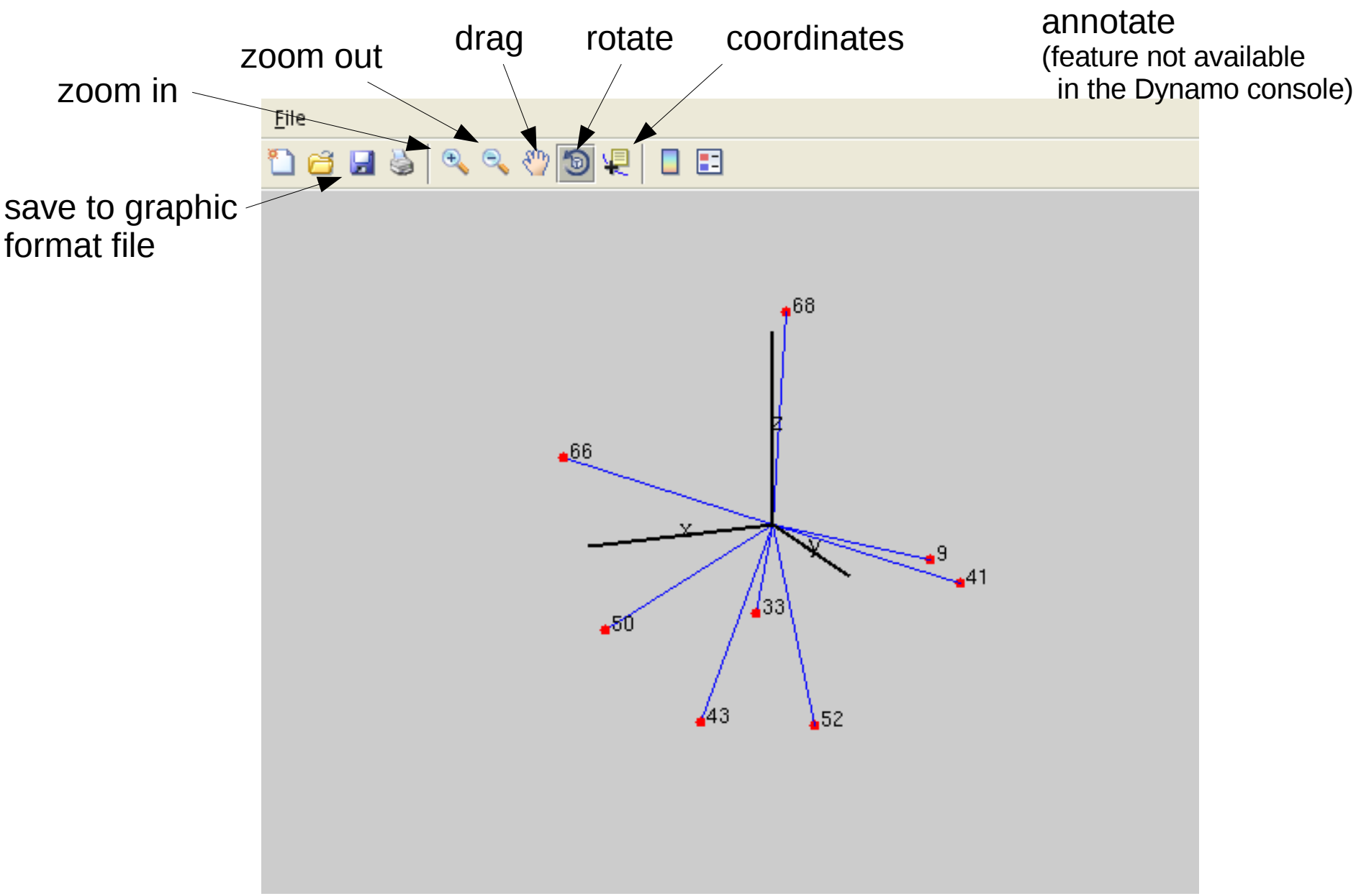
A clearer and more informative scene can be produced passing more parameters. Try:

```
dtplot test/real.tbl -lines on -color r -ct tag
```

Here, `ct` is the contraction of 'column_text', we tell tableplot to use the tags of the particles as text label accompanying each particle. You should get something like the plot in next slide.

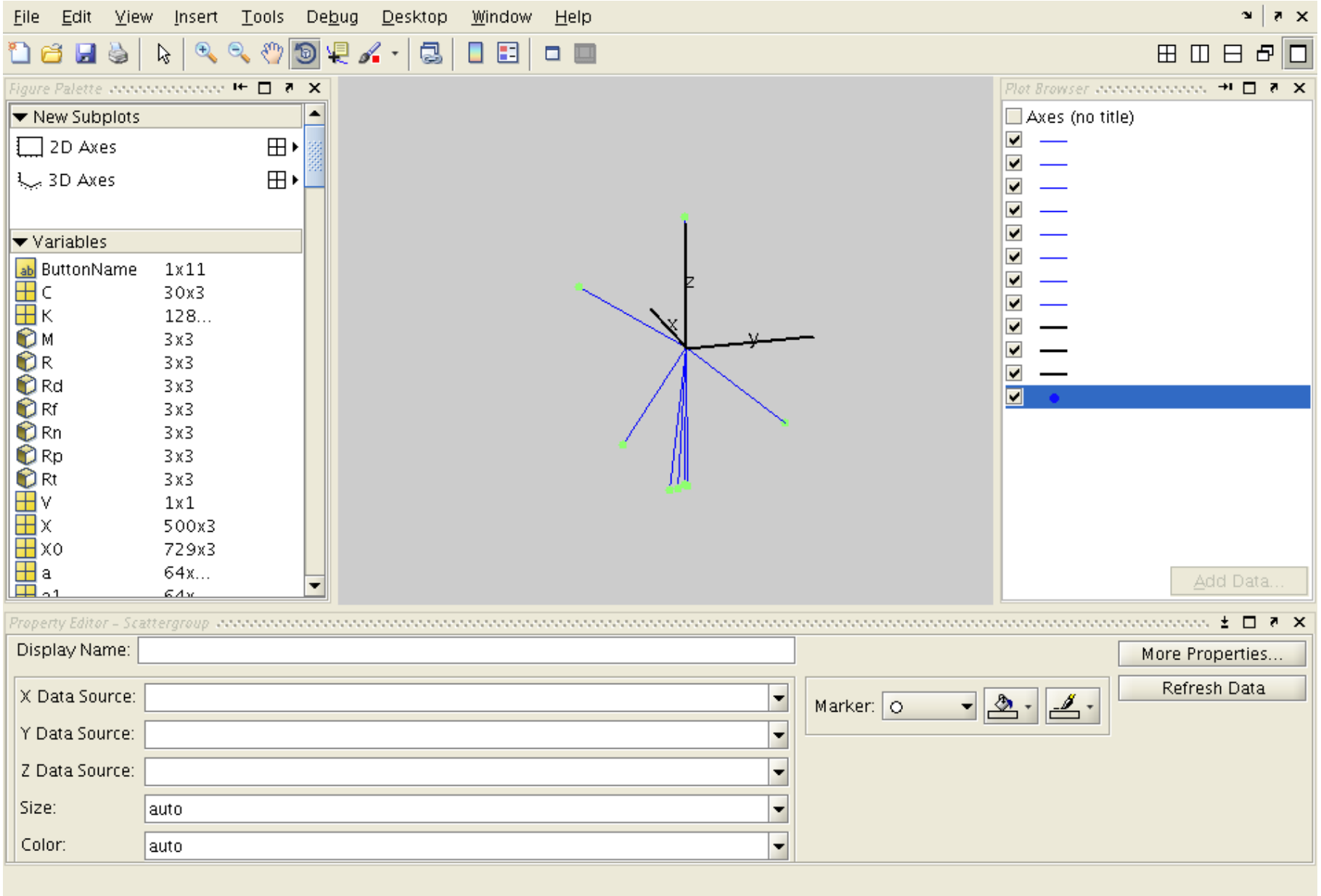
(you may need to close first the previous graphic to prevent depiction articafts, or choose a different window)

This is a normal Matlab graphical window, with its usual functionalities:



Note

if you are working in matlab, you have an additional tab for graphic edition of your scenes



`dtplot`: several depiction modii

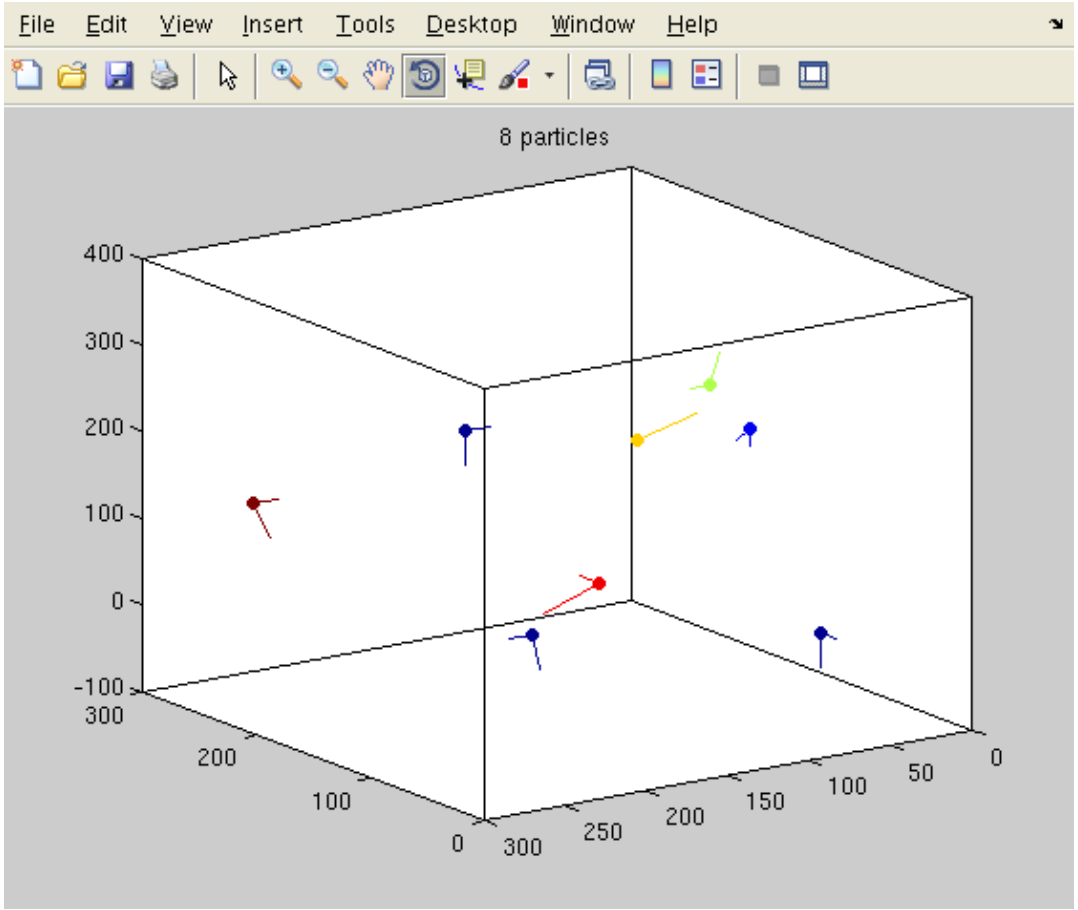
modus: '`sketch`': 3d positions of particles

```
dtplot test/real_table.tbl -m sk -sk 40 -c 8 -sm 30
```

length of each particle sketch

colours according to column 8 ('tilt')

marker size

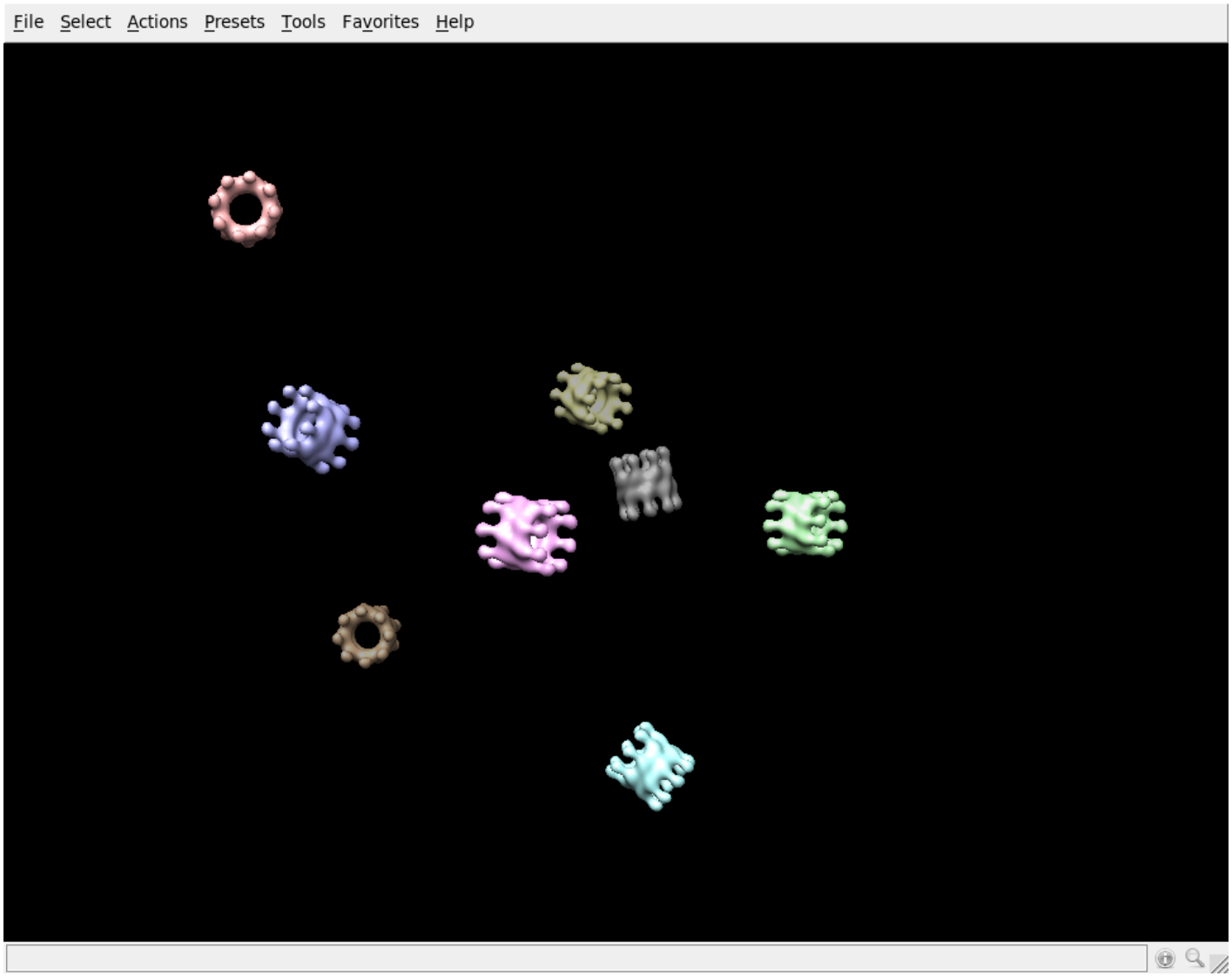


`dtplot` can also format data to pass angles and positions directly into Chimera:

```
dtplot test/real.tbl -m c -a inv -template test/original_template.em
```

modus: 'chimera'

actions: 'invert' (as chimera expects white protein on dark background)



an independent model
is opened for each particle

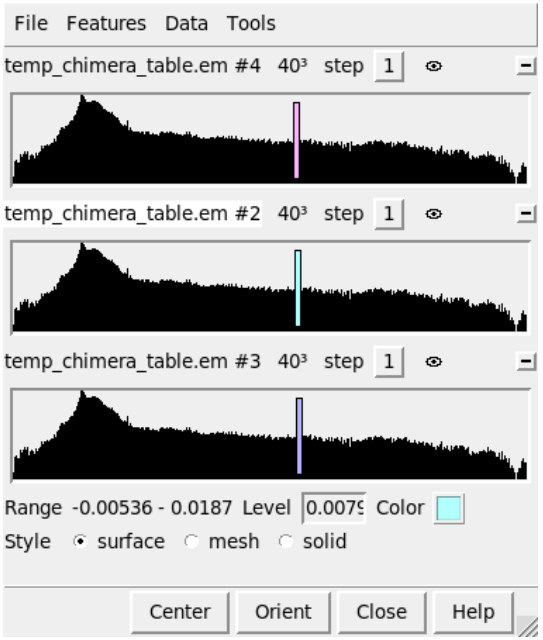


TABLE RESTRICTORS

Accessing particles that fulfill a given requirement or combination of requirements is a very common task.

Table restrictors are operators common to many Dynamo commands which perform this particle search inside a table on the fly.

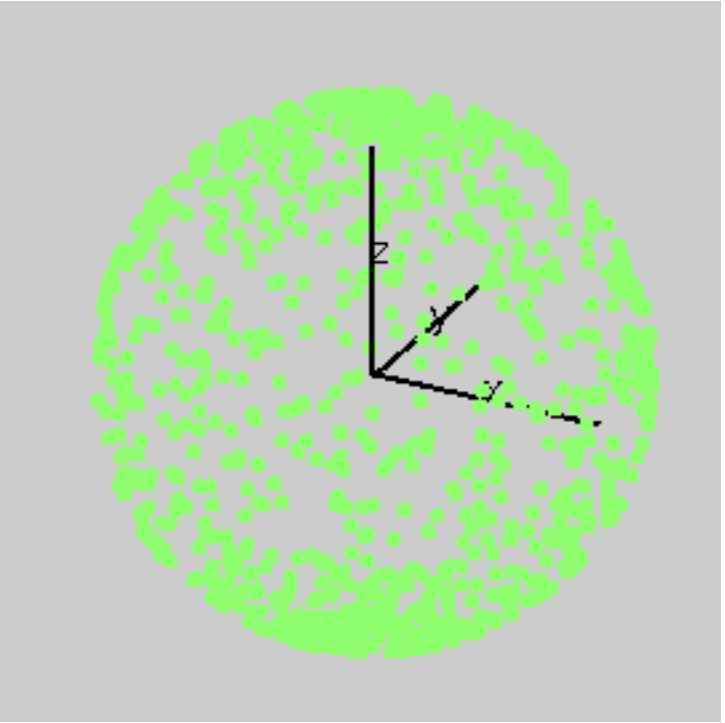
The explicit operator for restrictions is `dynamo_table_grep` (`dtgrep`) but in the next examples we will examine its action as auxiliary tool for other commands:

Let us create and example table with 1000 random orientations

```
>> dtrandom 1000 -o random.tbl
```

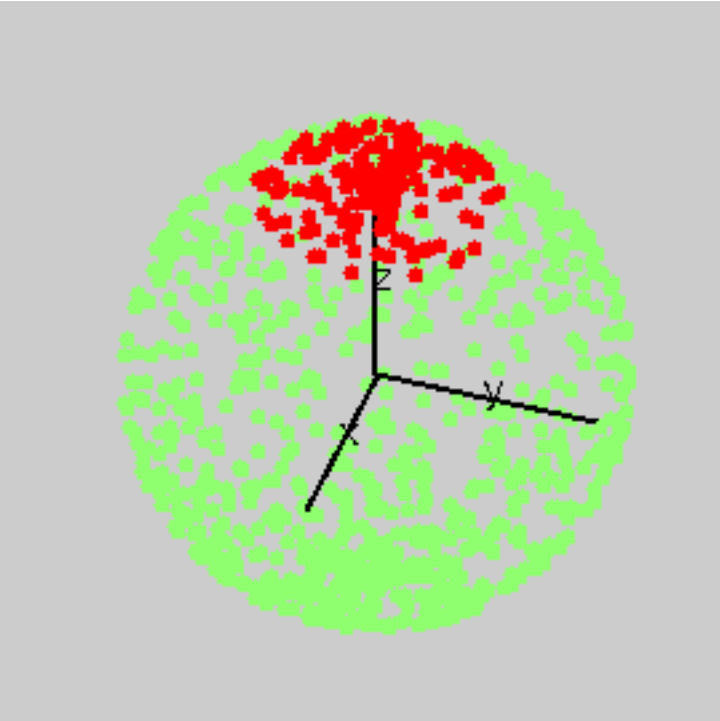
You can check that the generated particle orientations do cover the unit sphere:

```
>> dtplot random.tbl
```



Now we show the effect of table restrictors:

```
>> dtplot random.tbl -tr (abs)tilt<30 -color r
```



This was a rather simple, intuitive restriction, easy to depict.

Restrictors offer more posibilities:
* AND and OR operators,
* selection of angular directions
* use of functions

Access the help on `dtgrep` for a complete description

DATA

A further inhabitant of the `test` tutorial folder is a subfolder called `data`.

This is a *Dynamo*-style data folder, where particles are called following the convention:

```
test/data/particle_<tag>.em
```

with the tag number padded to five with zeros.

This format makes the folder recognizable for most Dynamo commands, and we can access it comfortably in different ways.

First, try the “info” command for data folders: dynamo_data_info. Type:

```
>> ddinfo test/data
```

You should get the results on the next slide, which provide an overview on the contents of the folder.

number of particles

sidelength

Session Edit View Bookmarks Settings Help

```
Dynamo >  >> ddinfo test/data
"ddinfo" references source command "data_info"

------------------------------------------------------------
  Executing:  data_info

Parsing 1 arguments
------------------------------------------------------------

              asproject: 0
            folder_name: 'test/data'
                correct: 1
                      N: 8
                   tags: [9 43 66 33 50 68 41 52]
                     Mb: 2
                      l: 64
            source_type: 'dynamo_folder'
              extension: 'em'
                padding: 5
                  exist: 1
                 Mb_all: 16
    N_accompanying_fmask: 0


   [ok]  data info

------------------------------------------------------------

  Exiting:  data_info
------------------------------------------------------------
```

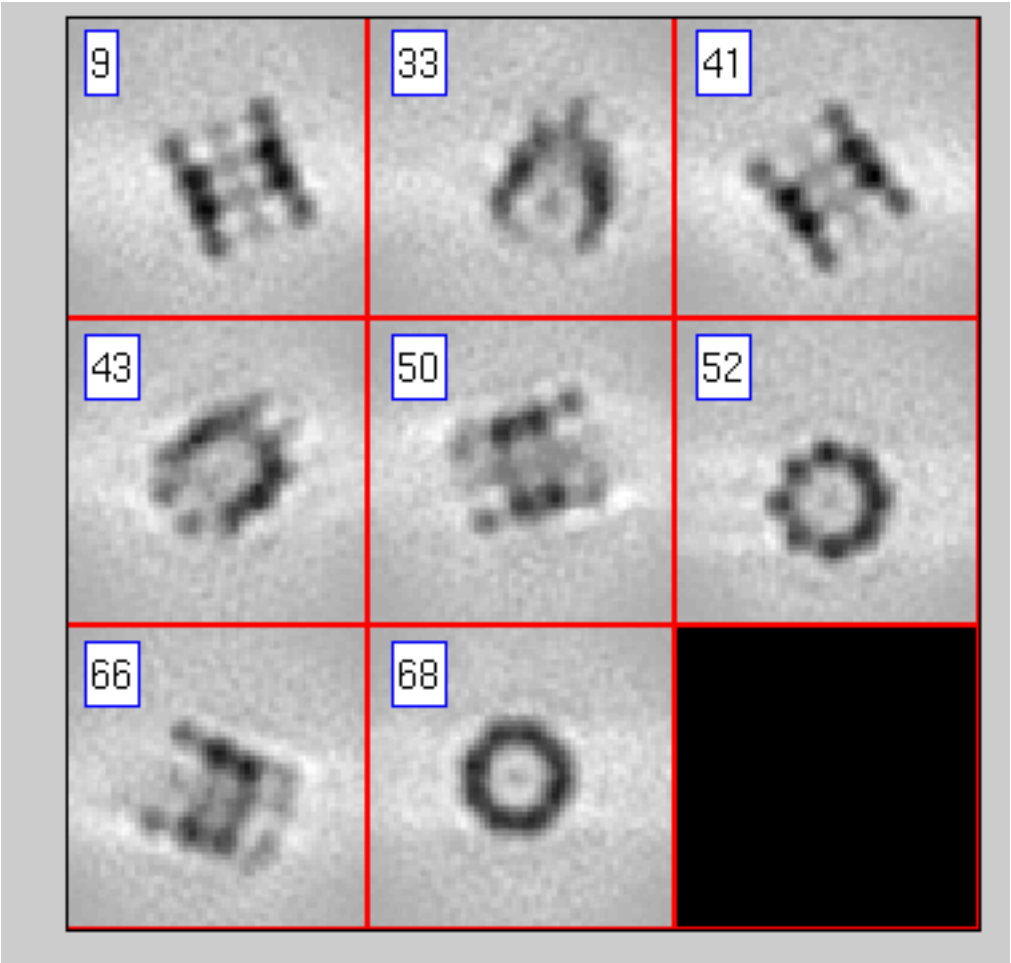
Shell

But they can get viewed directly, as a whole:

```
dslices test/data -labels tags -j c8 -ls 12
```

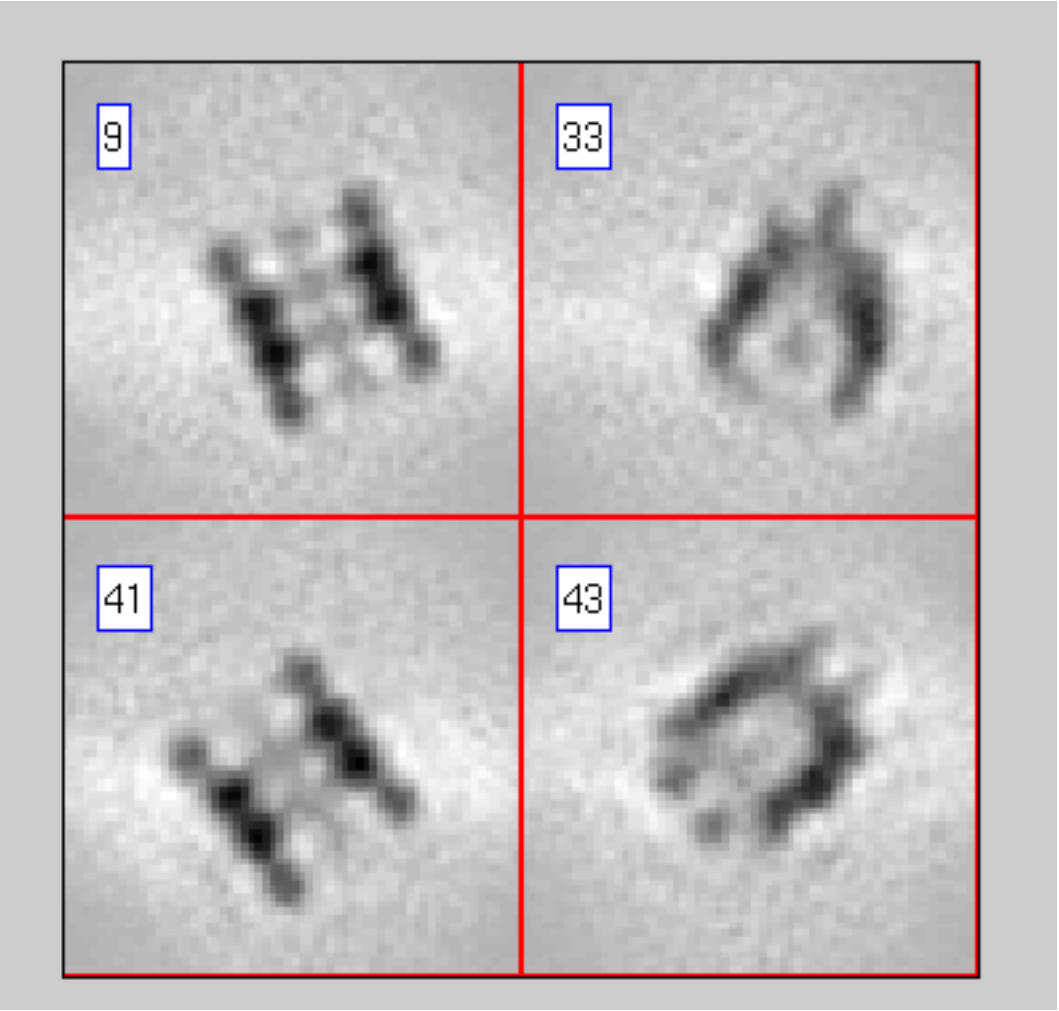
labels the “tag” number of the files

projects 8 slices from the center: (by default in direction z)



or for individual particles or user defined sets

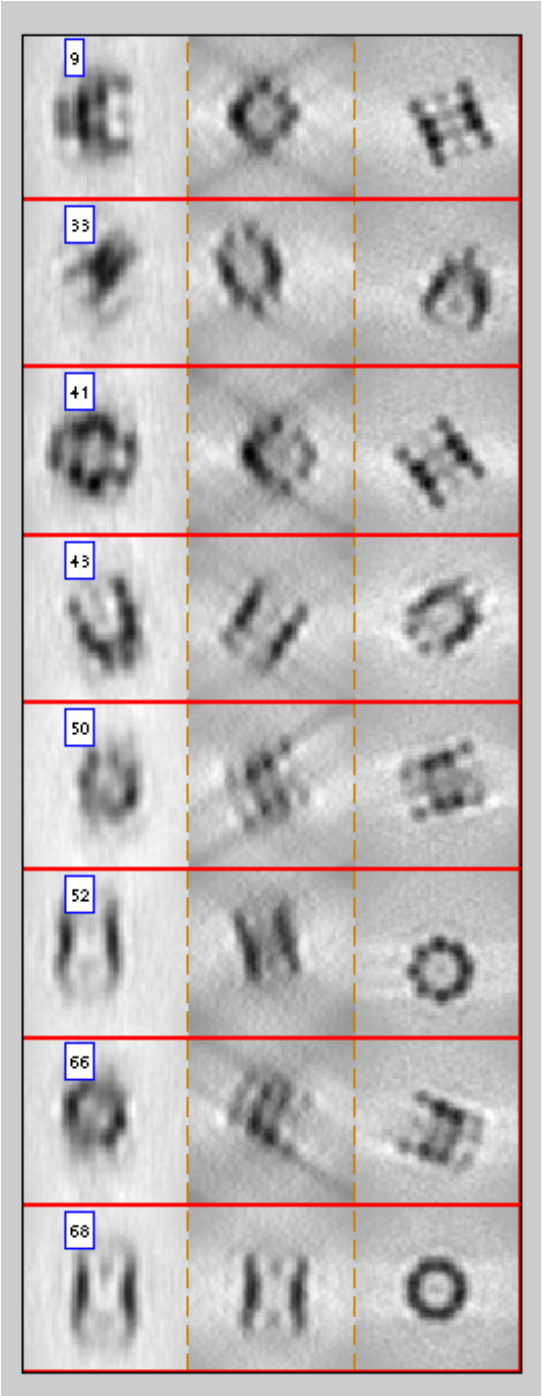
```
dslices test/data -labels tags -tags [9:43] -j c8 -ls 12
```



```
dslices test/data x|y|z -labels tags -j c8 -dim [8,1]
```

simultaneous
viewing directions

labels for each particle
(tags are deduced by
parsing the filenames)



TABLES AND DATA

Tables are important because they describe the geometry of data particles.

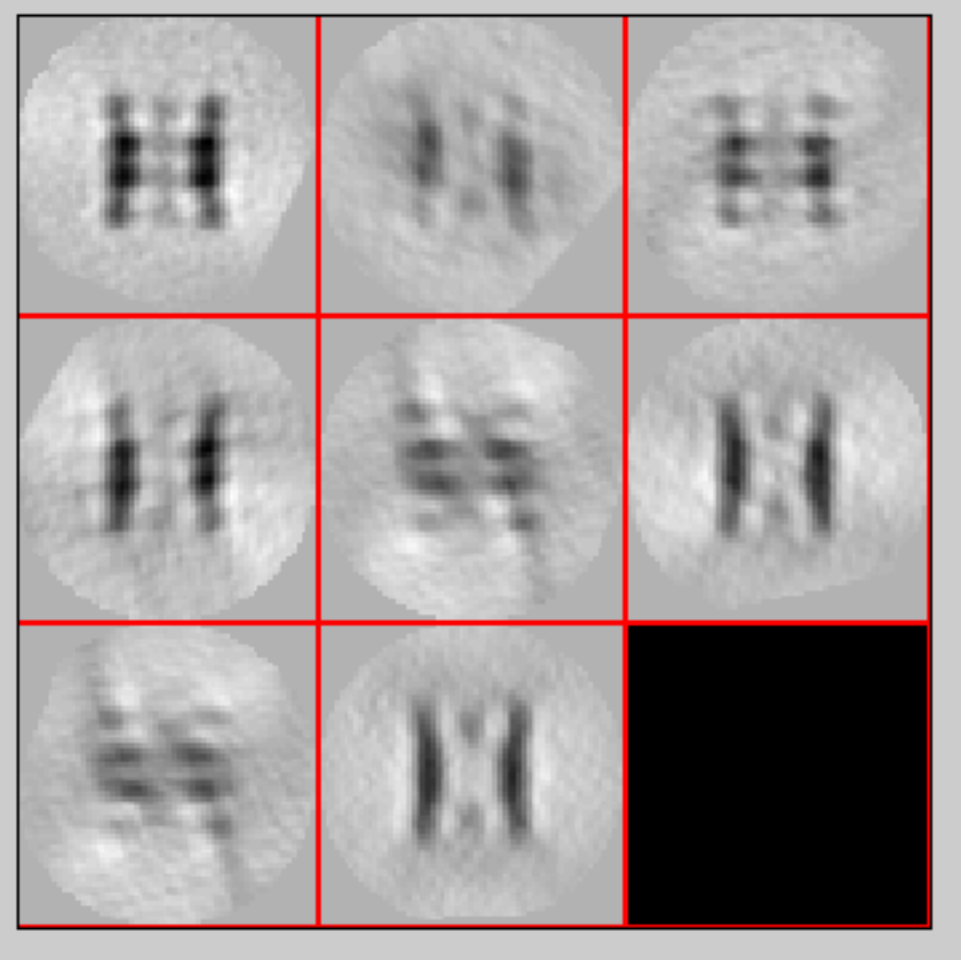
They are connected by the “tag” numbers that determine which row in a table refers to which particle file in the data set.

Let's see the basics on how tables and data sets work together.

We can pass directly a table:

```
>> dslices test/data -jy c6  -t test/real.tbl -align on;
```

and tell the command that it has to be used to align the particles



And again, table restrictors can be embedded into the command

```
dslices test/data -jy c6 -align on -t test/real.tbl  -tr [mang]0,0,0,60;
```

Here the restrictor [mang] (mirror + angle) selects all the particles around the angular direction [0,0,0] or its mirror direction with a maximum aperture of 60 degrees.

In this case, only two particles of the original two will survive.