



Subboxing: extraction of boxes out of subtomograms

What is “subboxing?”

We call “subboxing” the extraction of smaller subvolumes from inside the subtomograms, using the information about their location and orientation gained in previous alignment steps.

The set of extracted subvolumes can be treated as a new data set, potentially showing some advantages, mainly:

1) Computational speed

When particles are extracted from tomograms, they are typically cropped out with conservative criteria, as it is not always clear which are the actual extents of the particle. This might lead to unnecessary increases of the computational burden.

2) Extraction of assymmetric subunits

While symmetry imposition is useful during the first steps of an alignment, the capacity of working with the independent subunits might lead to fine improvements in the resolution, and also to the obtention of information that might ease the analysis of possible heterogeneities in the data.

In this first tutorial we will use subboxing just to reduce size of the data

```
>> dtutorial tsub
```

As we know, this creates (among other elements):

- * A template
- * A data set with randomly rotated versions of the template
- * A table that codify the alignment parameter of that data set

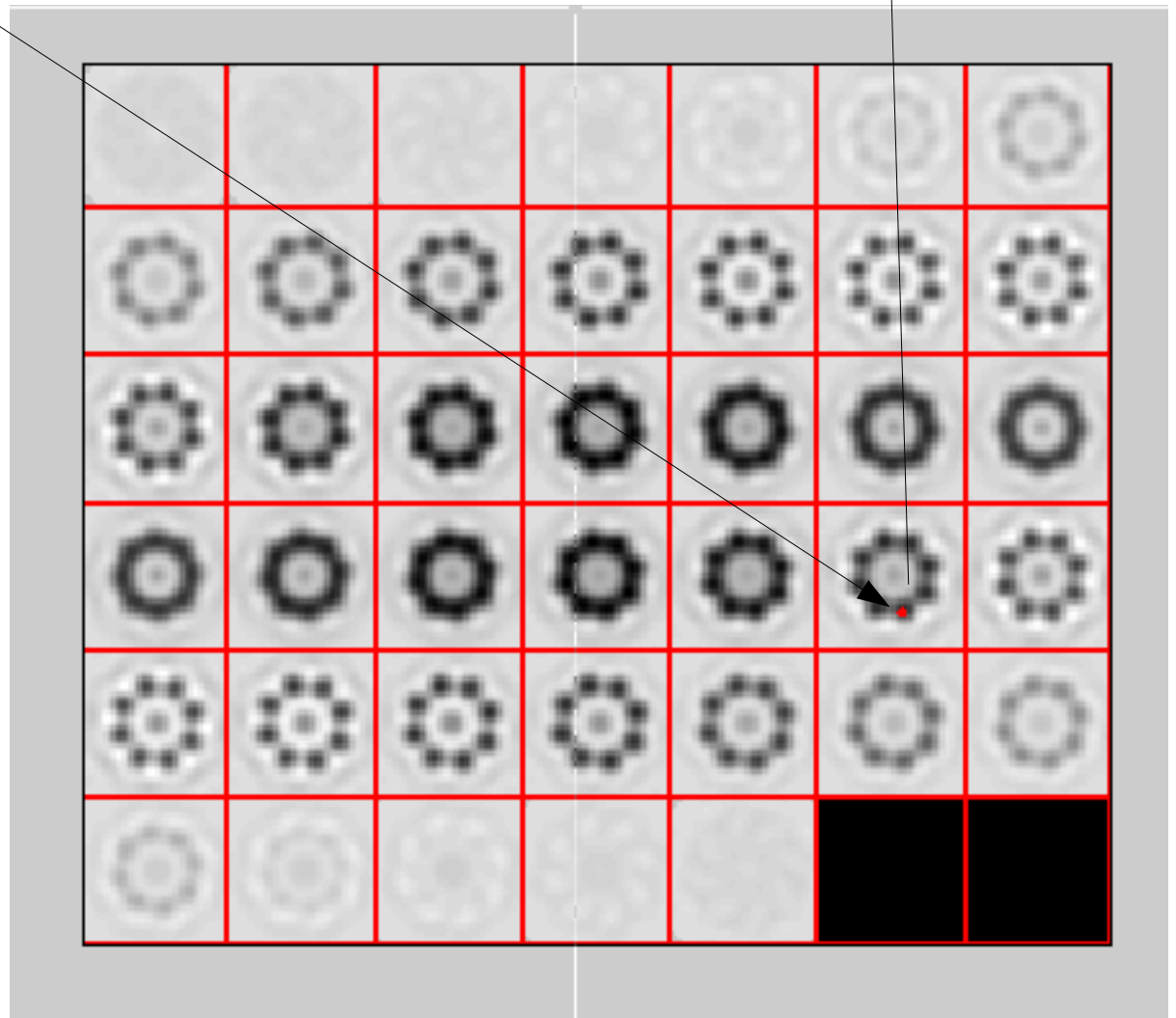
Let us choose the
coordinate with the mouse:

```
>> dslices tsub/original_template.em -click a;
```

Mouse buttons:

left click to select
middleclick to just show the 3d coordinates of the voxel (without s
right click to quit and accept selection

Matrix coordinates of selected point #1: 23.3 9.8 27.0



And now we create a “subboxing folder”:

subbox center as seen (or clicked) on the template

sidelength of the subbox

sidelength of the template
(if it differs from the
sidelength
of the data boxes)

table required to locate in each
particle the homologous position
to the one defined on the template
(the one passed with flag “-r”)

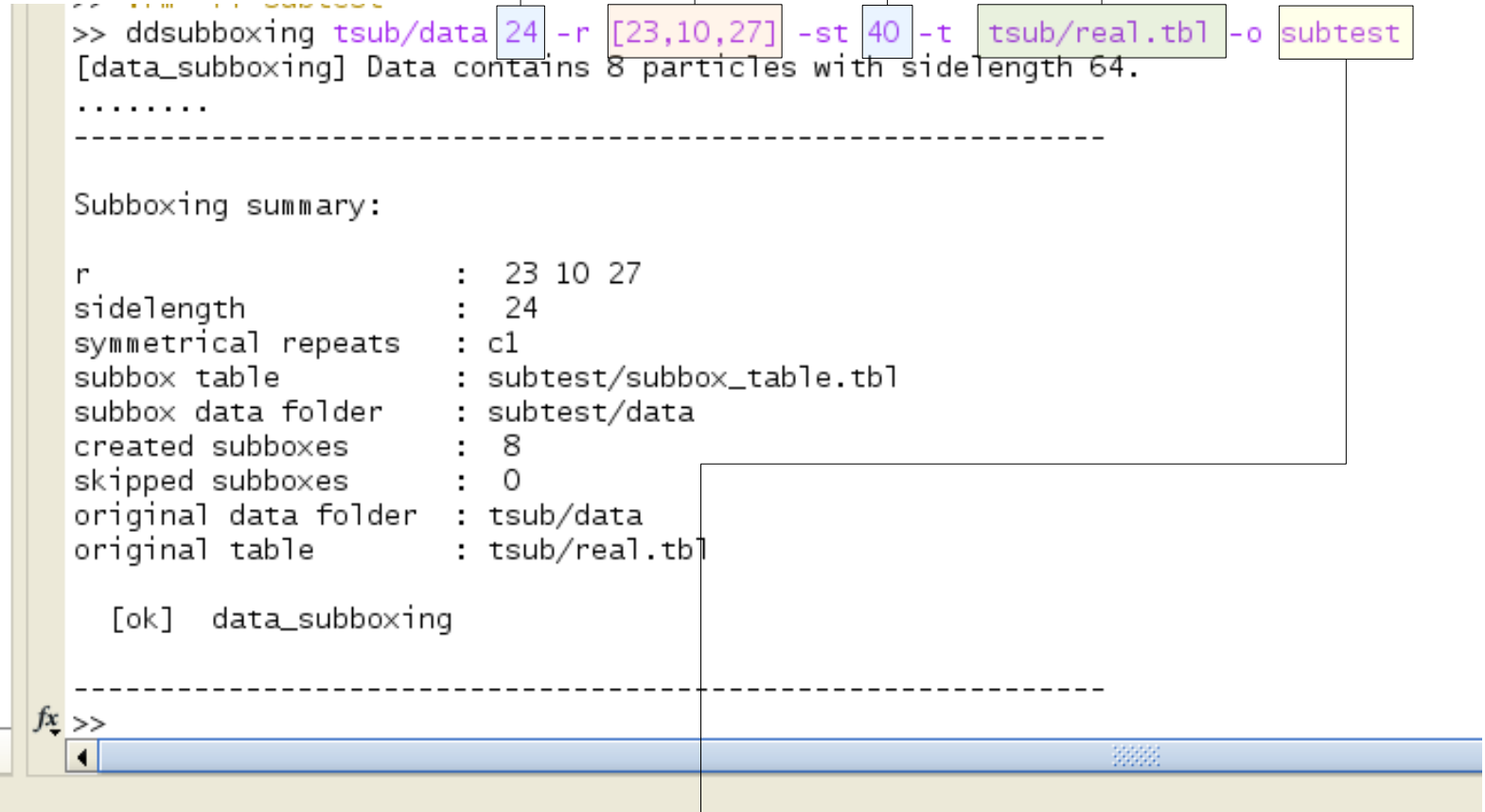
```
>> ddsubboxing tsub/data 24 -r [23,10,27] -st 40 -t tsub/real.tbl -o subtest
[data_subboxing] Data contains 8 particles with sidelength 64.
.....
-----

Subboxing summary:

r                : 23 10 27
sidelength       : 24
symmetrical repeats : c1
subbox table      : subtest/subbox_table.tbl
subbox data folder : subtest/data
created subboxes  : 8
skipped subboxes  : 0
original data folder : tsub/data
original table     : tsub/real.tbl

[ok] data_subboxing

-----
fx >>
```



An output folder will be created with different elements for later reference

```
>> !rm -rf subtest
>> ddsubboxing tsub/data 24 -r [23,10,27] -st 40 -t tsub/real.tbl -o subtest
[data_subboxing] Data contains 8 particles with sidelength 64.
```

```
.....
```

```
-----
Subboxing summary:
```

```

r                : 23 10 27
sidelength       : 24
symmetrical repeats : c1
```

```
subbox table      : subtest/subbox_table.tbl
subbox data folder : subtest/data
```

```
created subboxes  : 8
skipped subboxes   : 0
```

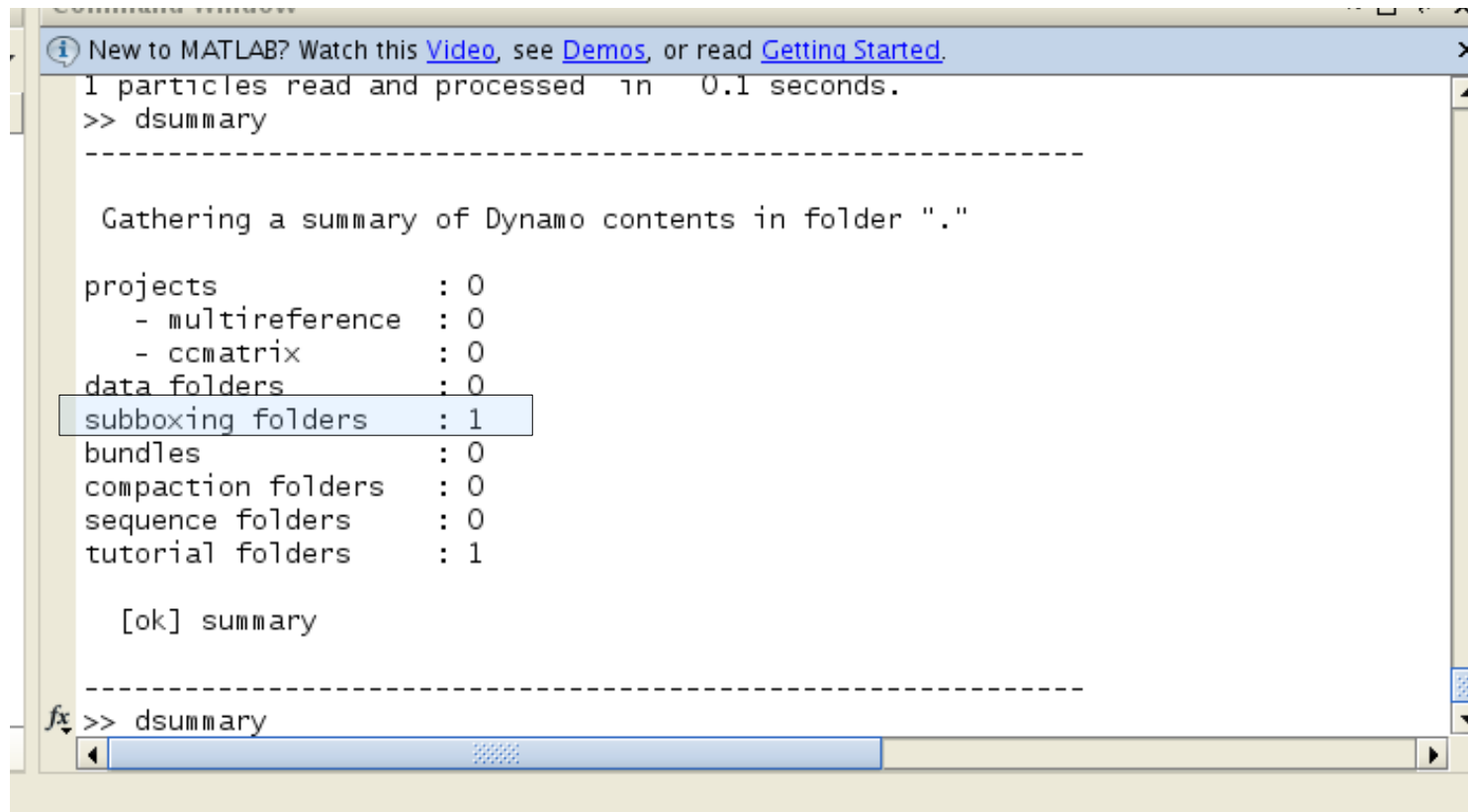
```
original data folder : tsub/data
original table       : tsub/real.tbl
```

```
[ok] data_subboxing
-----
```

In the folder subtest, you have thus a new description of your data in terms of a data set (with cropped subparticles) and a table (that expresses the old table adapted to the recentering of the particles)

just for bookkeeping on how the subboxing folder was constructed

Note that the subboxing is one of the items registered by `dsummary`



The screenshot shows a MATLAB Command Window. At the top, there is a blue header bar with a question mark icon and the text: "New to MATLAB? Watch this [Video](#), see [Demos](#), or read [Getting Started](#)." Below this, the command window shows the execution of the `dsummary` command. The output indicates that 1 particle was read and processed in 0.1 seconds. The command `>> dsummary` is entered, followed by a dashed line separator. The output text "Gathering a summary of Dynamo contents in folder \".\" is displayed. A list of items follows: "projects : 0", "- multireference : 0", "- ccmatrix : 0", "data folders : 0", "subboxing folders : 1", "bundles : 0", "compaction folders : 0", "sequence folders : 0", and "tutorial folders : 1". The "subboxing folders : 1" line is highlighted with a blue selection box. Below the list, the text "[ok] summary" is shown. Another dashed line separator follows. At the bottom, the command prompt `fx >> dsummary` is visible, with a blue selection box highlighting the command.

```
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
1 particles read and processed in 0.1 seconds.
>> dsummary
-----

Gathering a summary of Dynamo contents in folder "."

projects          : 0
- multireference  : 0
- ccmatrix        : 0
data folders      : 0
subboxing folders : 1
bundles           : 0
compaction folders : 0
sequence folders  : 0
tutorial folders  : 1

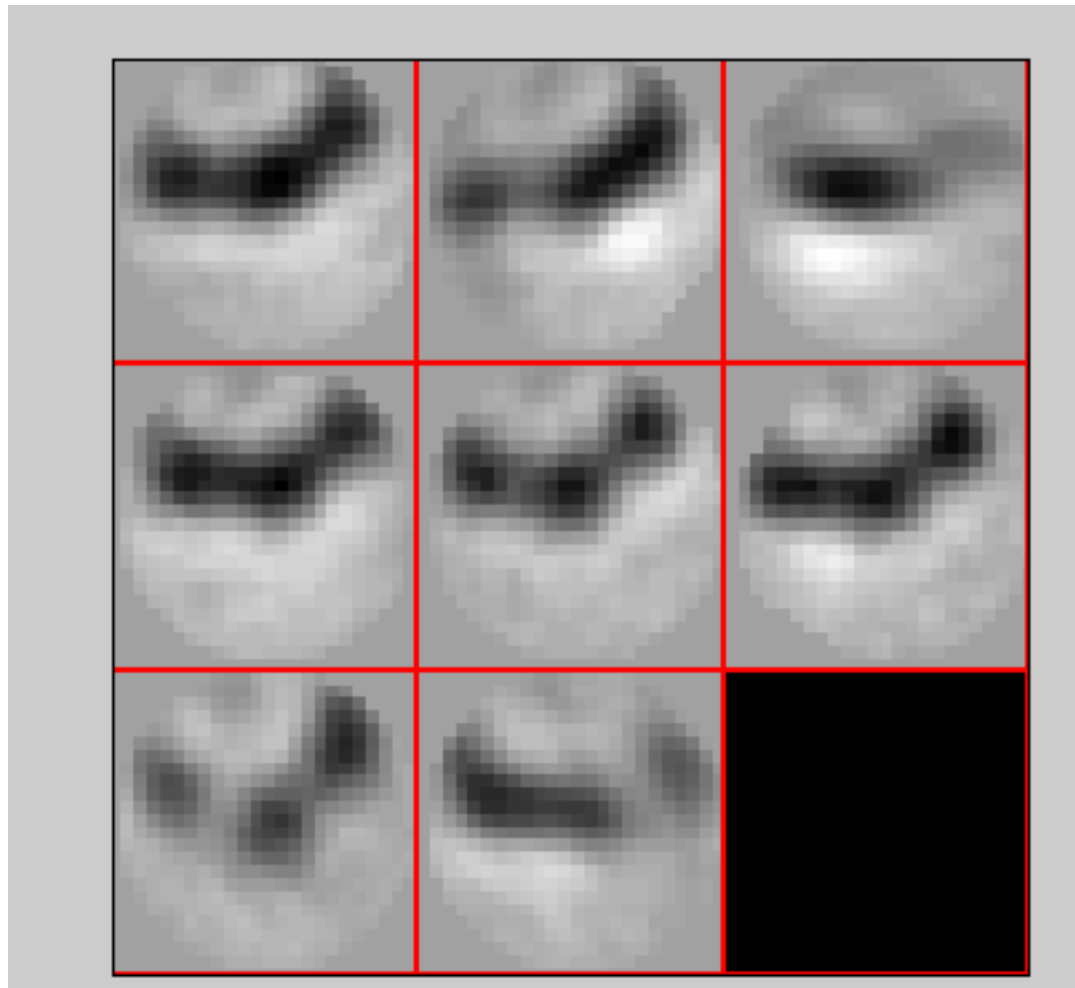
[ok] summary
-----
fx >> dsummary
```

(a higher verbosity would reveal the actual names)

we check now the contents of this subboxing directory

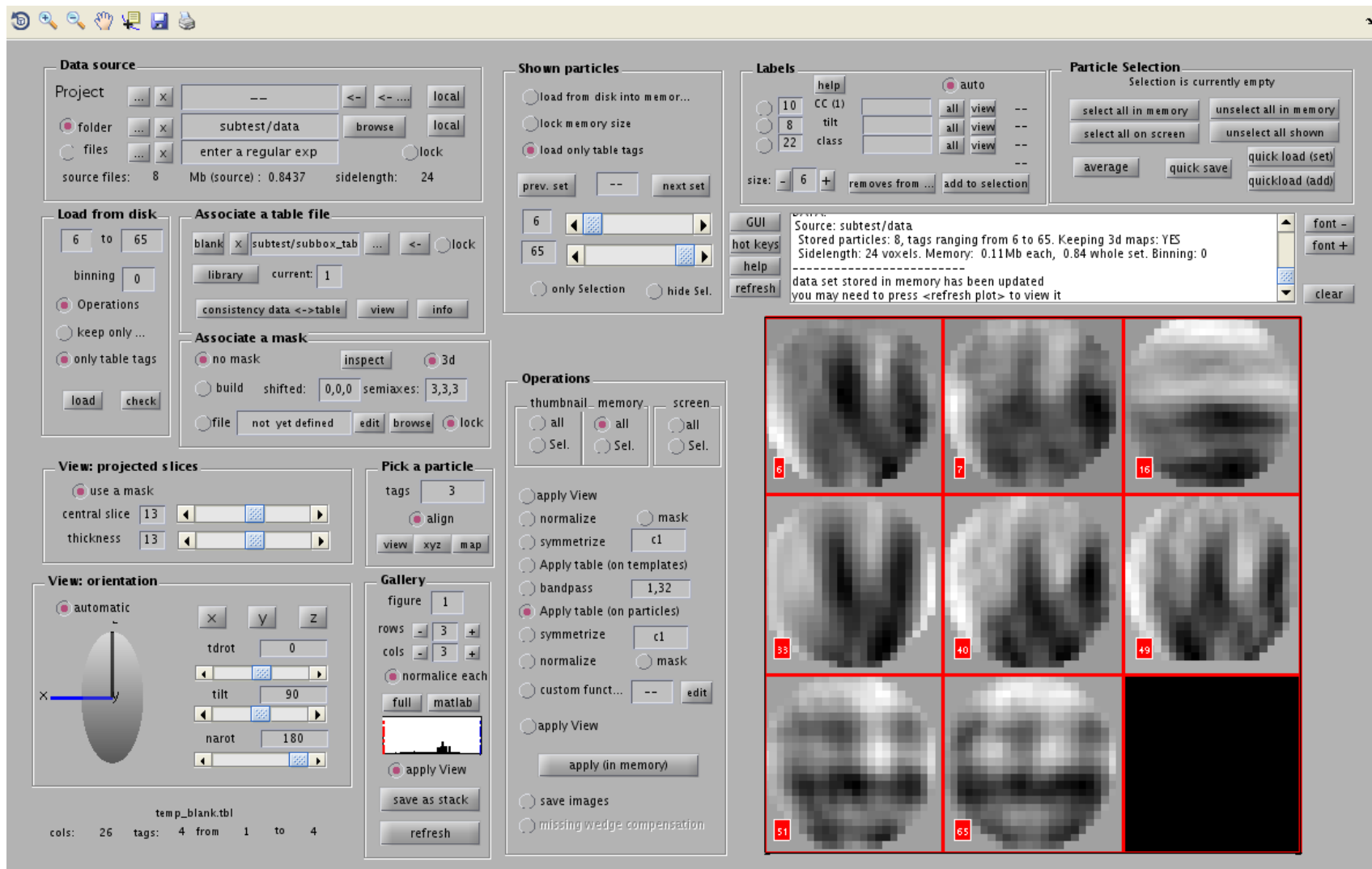
```
dslices subtest/data z -j c0 -t subtest/subbox_table.tbl -align on
```

We use the table generated by Dynamo referred to the cropped subboxes



The particles are now smaller and centered on the spot we clicked.

They don't look exactly equal as each one has a different missing wedge.



dgallery -d subtest/data -t subtest/subbox_table.tbl -load on;

Now that we know how (simple) subboxing works, we can go further and sketch a further step, where we see how “subboxing” interacts with

```
>> dtutorial t2 -p pt2
```

This project will be used to create a coarse approximative solution to the problem in the dataset, which will be later refined by subboxing.

Already in this project, before doing any subboxing operation, we

As it is a minor modification of the project, we do it with the light tool dvput, instead of opening dynamo_project_manager

1 – modify ite in round 2 ('ite_r2') to 0

2 – check, save, unfold

The screenshot shows the DVPUT software interface. On the left, the 'Project' panel displays buttons for 'check', 'save', 'unfold', and 'run'. Below it, the 'Info' panel shows a message: 'loading settings of project "pt2"' and 'project loaded'. At the bottom left, a table lists project parameters and their values.

Parameter	Value
file_template_initial	t2/template.em
file_table_initial	t2/initial.tbl
folder_data	t2/data
file_mask	t2/mask.em
file_mask_classification	t2/mask_classification
file_fmask_initial	t2/fmask.em
destination	system_omp
how_many_processors	1
cluster_header	cluster_header.sh
cluster_walltime	00:10:00
submit_order	sbatch
gpu_identifier_set	0

On the right, a large table displays parameters for rounds 1 through 6. The 'ite' parameter in round 2 is highlighted with a blue box, indicating it has been modified to 0.

	round 1	round 2	round 3	round 4	round 5	round 6	rou
ite	1	0	0	0	0	0	0
nref	1	1	1	1	1	1	1
cone_range	360	60	360	360	360	360	360
cone_sampling	60	20	45	45	45	45	45
inplane_range	60	20	360	360	360	360	360
inplane_sampling	20	5	45	45	45	45	45
high	0	0	2	2	2	2	2
low	21	32	32	32	32	32	32
sym	c1	c8	c1	c1	c1	c1	c1
dim	32	64	0	0	0	0	0
refine	6	6	6	6	6	6	6
refine_factor	2	2	1.80	1.80	1.80	1.80	1.80
area_search	1 1 1	1 1 1	4 4 4	4 4 4	4 4 4	4 4 4	4 4 4
area_search_modus	0	0	0	0	0	0	0
use_CC	1	1	1	1	1	1	1
localnc	1	1	1	1	1	1	1
threshold	0.20	0.20	0.20	0.20	0.20	0.20	0.20
threshold_modus	0	0	0	0	0	0	0
threshold2	0.20	0.20	0.20	0.20	0.20	0.20	0.20
threshold2_modus	0	0	0	0	0	0	0
ccmatrix	0	0	0	0	0	0	0
ccmatrix_type	bin 0; sy...	sym c8	align	align	align	align	align
ccmatrix_batch	4	4	128	128	128	128	128
Xmatrix	0	0	0	0	0	0	0
Xmatrix_maxMb	100	100	100	100	100	100	100
PCA	0	0	0	0	0	0	0
PCA_neigs	4	4	4	4	4	4	4
kmeans	0	0	0	0	0	0	0
kmeans_ncluster	2	2	2	2	2	2	2
kmeans_ncoefficients	3	3	3	3	3	3	3
nrlacc	1	1	1	1	1	1	1

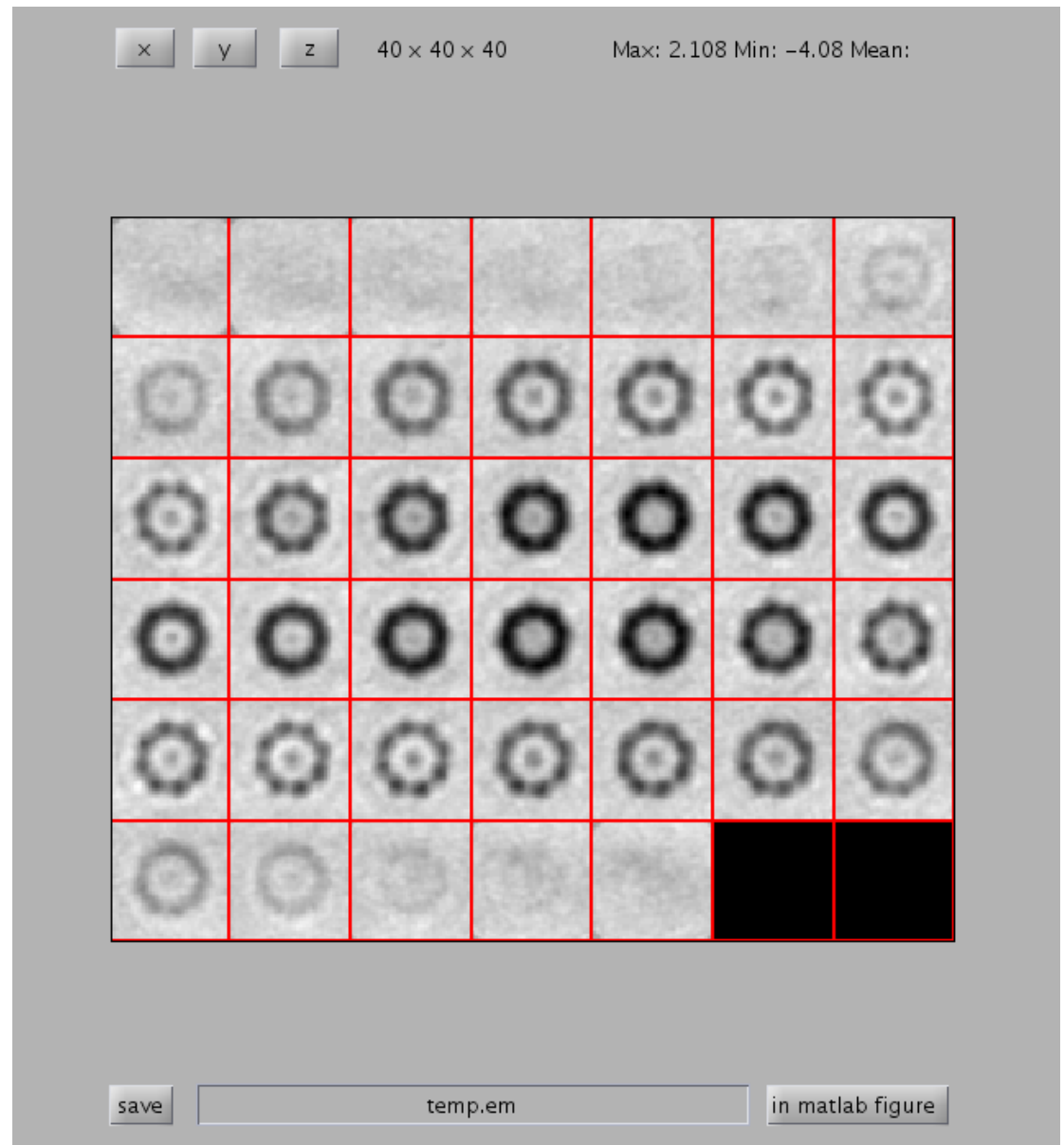
Note that this is equivalent to simply typing on the command line:

```
>> dvput pt2 unfold -ite_r2 0;
```

We are just computing a coarse orientation using one single iteration.

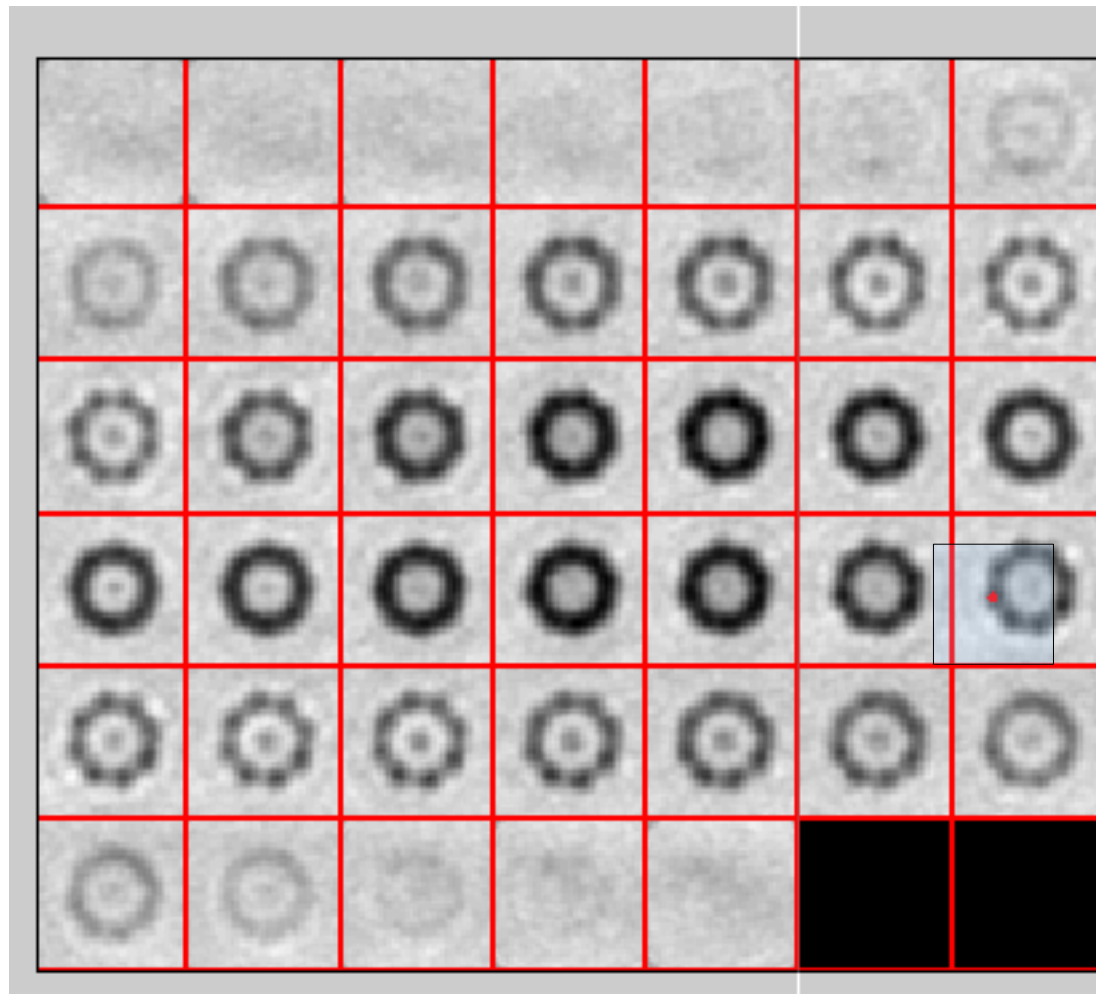
It will thus end very soon, and we can quickly check that the average that we get is decent enough as first approximation:

```
ddb pt2:a -v
```



And we now will set a new project on this by creating a “subboxing” folder as we just saw:

```
>> dslices pt2:a -click r
```



coordinates 11,17,28

And we create a subboxing directory based on the data and table of the “coarse” project, centering the particles on the point that we just clicked:

```
ddsubboxing pt2:data 24 -r [11,17,28] -st 40 -t pt2:t:ite=1 -o subunits;
```

just a database query for the data folder

Note that we need to pass the size of the template, as in this tutorial data and template are of different size (default setting of the tutorial generation)

and also a database query to access directly the last (and in this case unique) table in the folder

Now we enter the subboxing folder we just created and we will proceed from there:

```
cd subunits;
```

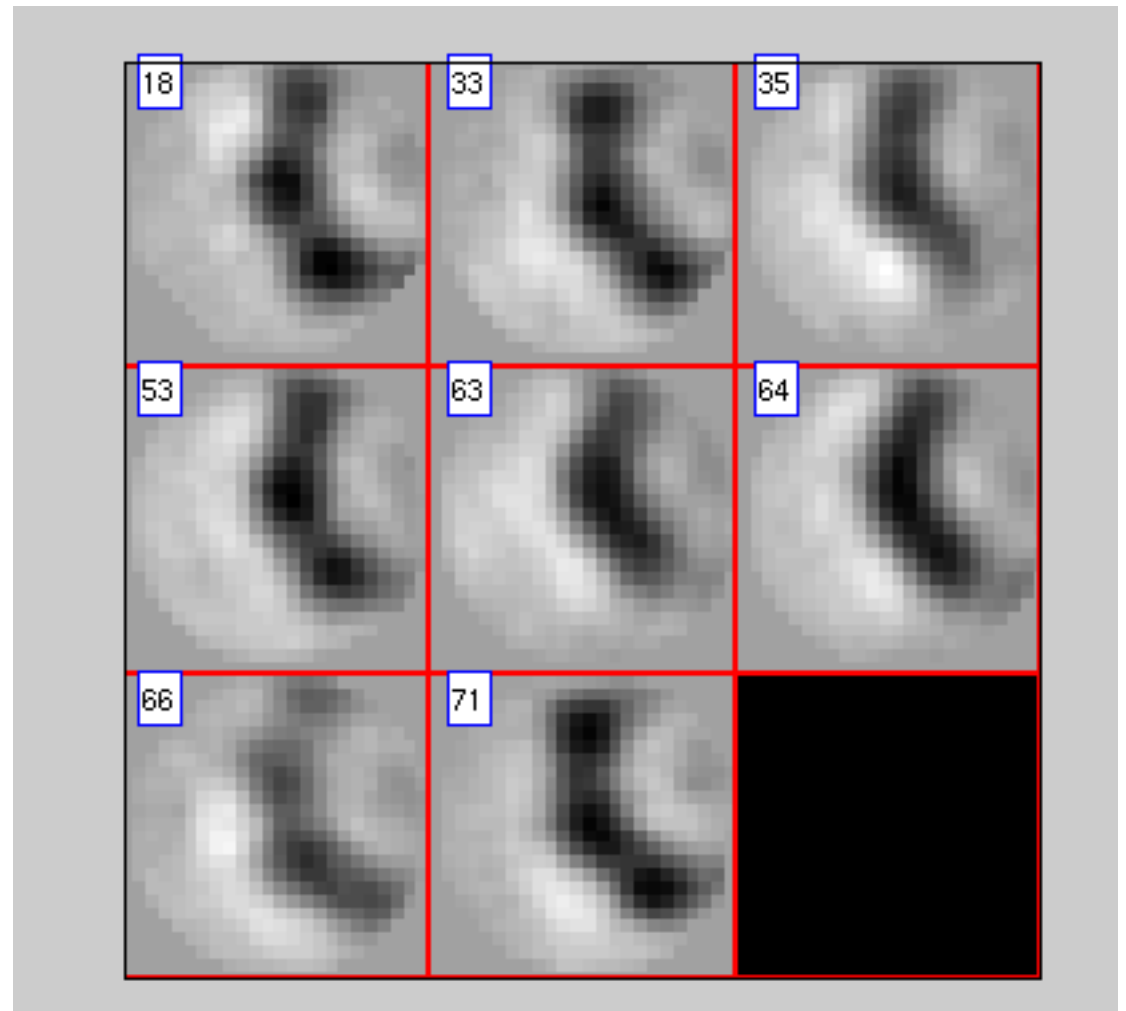
Perhaps a check that everything is as we expected it:

```
dslices data z -proj * -t subbox_table.tbl -align on -labels tags
```

looks good...

so, we can now create a project that refines the alignment on this data.

You can do that with the tools that you already know (project_manager, dvput), but let us see how to do it from the command line and using some tools foreseen for this subboxing technique:



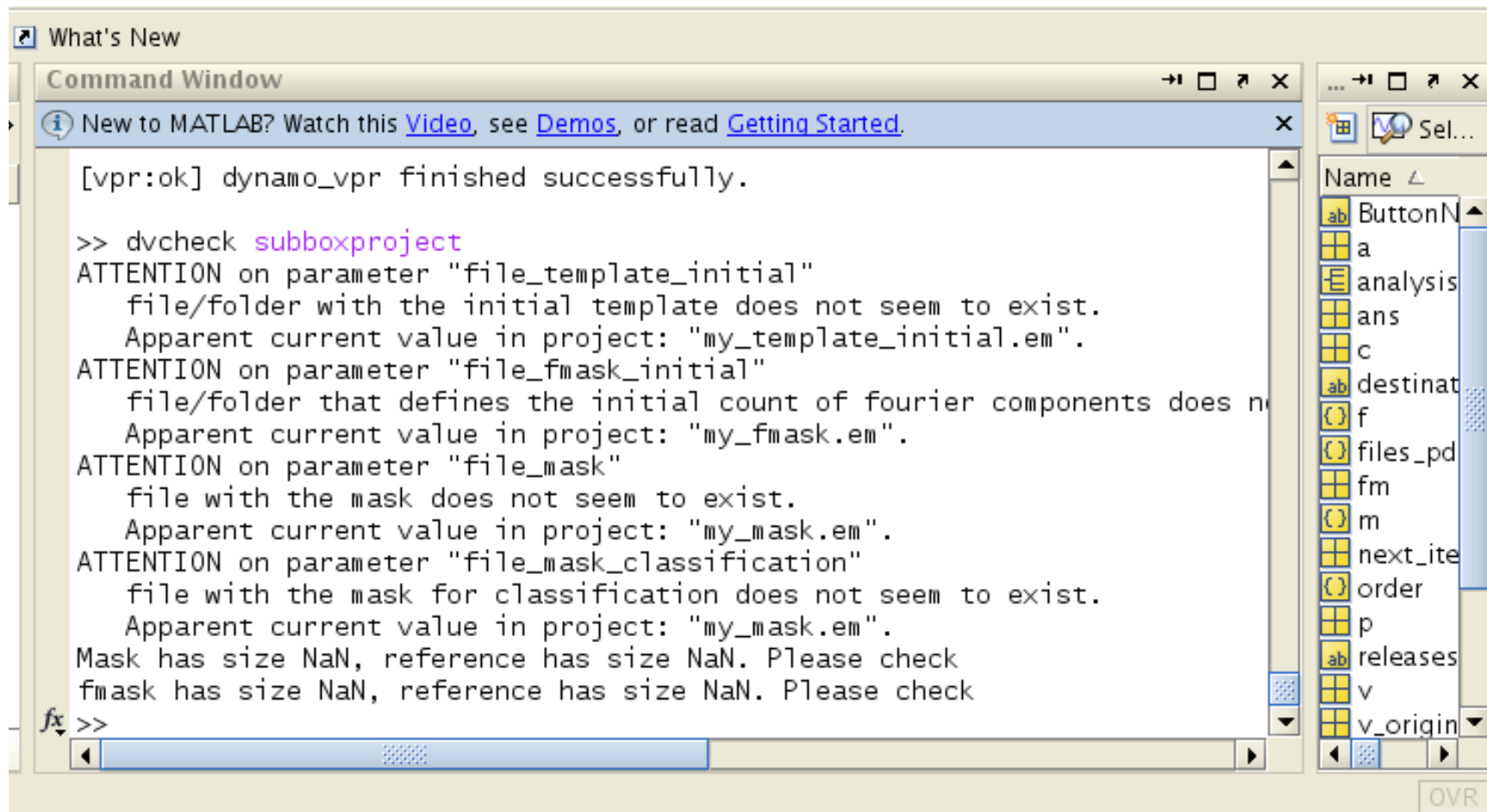
We can, for instance, create a project ab initio, using the table and the data that we have

```
dv subboxproject -d data -hint refine -s d -t subbox_table.tbl
```

generates numerical settings for refinement

saves the project in disk, but does not unfold it [this is actually the default]

Of course, the rest of the files (template, masks,...) are missing, and if we check the project we will get a full list of complaints:



To populate them, we define this project as refinement project of the original source project... what actually is!

```
dvsources ../pt2 subboxproject -modus update -import volumes;
```


This order populates the new project subboxproject with the adequate entries of the source project

```
Target project needs 1 multireference channel(s).

-----
Importing "mask" from source  to be used as "mask" in target

    source: "mask" ite:1 ref:1 --> target: "mask" ref:1
    ... copied

-----
Importing "mask_classification" from source  to be used as "mask_classification" in target

    source: "mask_classification" ite:1 ref:1 --> target: "mask_classification" ref:1
    ... copied

-----
Importing "average" from source  to be used as "template_initial" in target

    source: "average" ite:1 ref:1 --> target: "template_initial" ref:1
    ... copied

-----
Importing "fmask_average" from source  to be used as "fmask_initial" in target

    source: "fmask_average" ite:1 ref:1 --> target: "fmask_initial" ref:1
    ... copied

SUMMARY: Project parameters that will change in the target project to point to the new files.
    file_mask: './subboxproject/settings/mask.em'
    file_mask_classification: './subboxproject/settings/mask_classification.em'
    file_template_initial: './subboxproject/settings/template_initial_ref_001.em'
    file_fmask_initial: './subboxproject/settings/fmask_initial_ref_001.em'

[ok] vpr_source finished for target project "subboxproject"

fx >>
```

Note that dvsource is not only applicable in the subboxing settings, but it is a very general tool

We are coming closer to a project that we can execute!
But still we cannot:

forces to run a check before unfolding the project

```
>> dvunfold subboxproject -c on;  
reference has size 40, bigger than particle size 24. Please check  
%%% ERROR in intended project "subboxproject". Faulty virtual project %  
[vunfold] Fatal error found. Aborting the unfold of this project.
```

We have cropped the data, and now the template inherited from the coarse project is too big!

This is just because we used a generic tool for the “sourcing” of the old project... but it's not a problem, we just inform the project subboxproject that it is a project in a subboxing directory:

we can actually go for a project for very local refinement

with just one round

Project

... x load subboxproject local

check save unfold run

Info

font- font+ customize fonts reset message area

saving settings into project "subboxproject"

GUI settings entered into project. Project was not u

saving settings into project "subboxproject"

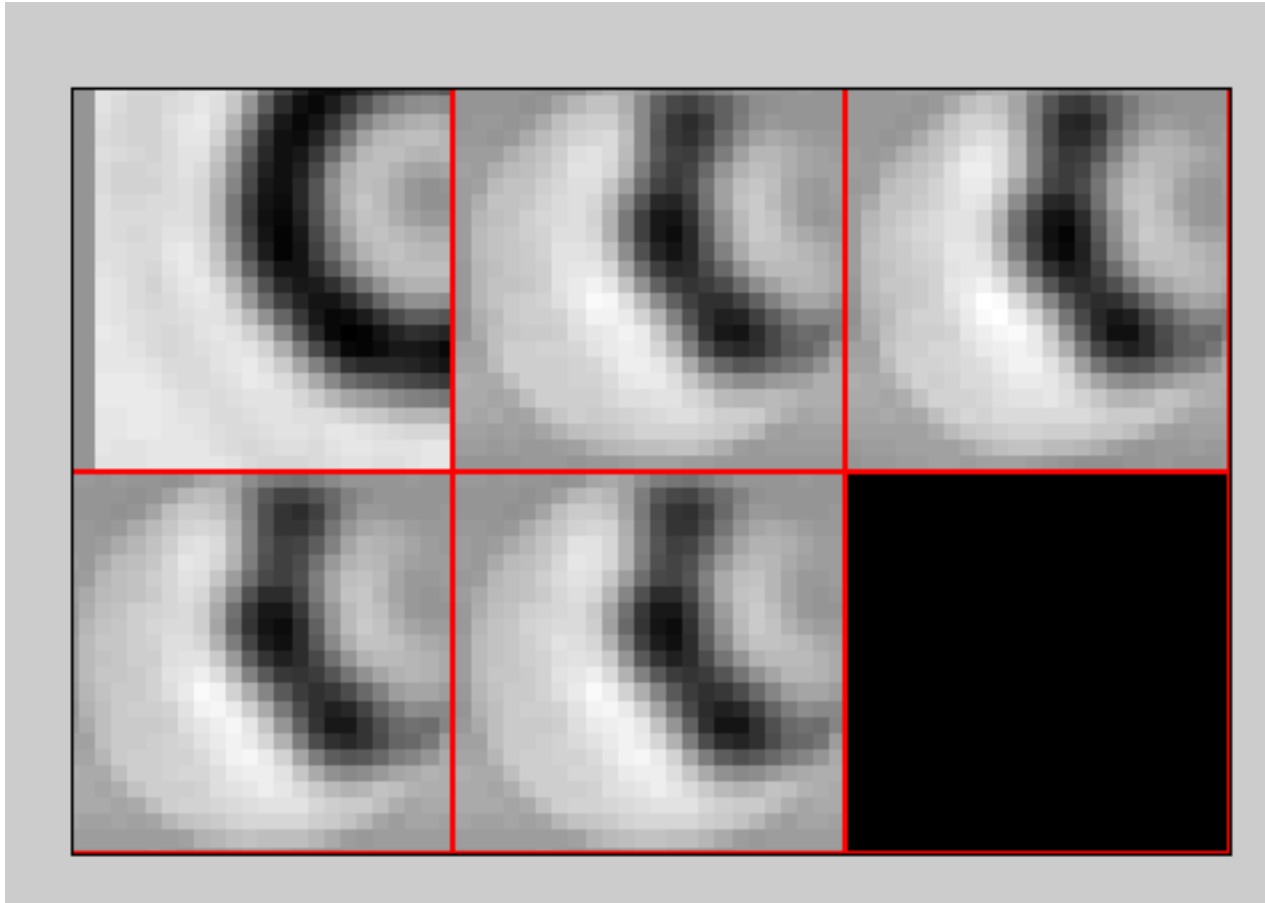
done

	1
file_template_initial	./subboxproject/se
file_table_initial	./subboxproject/se
folder_data	data
file_mask	./subboxproject/se
file_mask_classification	./subboxproject/se
file_fmask_initial	./subboxproject/se
destination	matlab
how_many_processors	1
cluster_header	cluster_header.sh
cluster_walltime	00:10:00
submit_order	sbatch
gpu_identifier_set	0

	round 1	round 2	round 3	round 4	round 5	round 6	rou
ite	4	0	0	0	0	0	0
nref	1	1	1	1	1	1	1
cone_range	6	20	10	5	1	360	360
cone_sampling	2	10	5	2	0.50	45	45
inplane_range	6	20	10	5	1	360	360
inplane_sampling	2	10	5	2	0.50	45	45
high	1	1	1	1	1	1	1
low	6	8	8	12	12	12	12
sym	c1	c1	c1	c1	c1	c1	c1
dim	24	24	24	24	24	24	24
refine	6	6	6	6	6	6	6
refine_factor	1.80	1.80	1.80	1.80	1.80	1.80	1.80
area_search	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	2 2 2	4 4 4
area_search_modus	1	1	1	2	2	2	2
use_CC	1	1	1	1	1	1	1
localnc	1	1	1	1	1	1	1
threshold	0.20	0.20	0.20	0.20	0.20	0.20	0.20
threshold_modus	0	0	0	0	0	0	0
threshold2	0.20	0.20	0.20	0.20	0.20	0.20	0.20
threshold2_modus	0	0	0	0	0	0	0
ccmatrix	0	0	0	0	0	0	0
ccmatrix_type	align	align	align	align	align	align	align
ccmatrix_batch	128	128	128	128	128	128	128
Xmatrix	0	0	0	0	0	0	0
Xmatrix_maxMb	100	100	100	100	100	100	100
PCA	0	0	0	0	0	0	0
PCA_neigs	4	4	4	4	4	4	4
kmeans	0	0	0	0	0	0	0
kmeans_ncluster	2	2	2	2	2	2	2
kmeans_ncoefficients	3	3	3	3	3	3	3
nclass	1	1	1	1	1	1	1

now, the project will run very rapidly, as it is defined as a local refinement, and the boxes are small (24 pixels).

As expected, it refines the structure around the asymmetric unit.



```
>>ddb subboxproject:average:ite=[0:4] -jz *
```