EDF R&D

Fluid Dynamics, Power Generation and Environment Department
Single Phase Thermal-Hydraulics Group

6, quai Watier
F-78401 Chatou Cedex

Tel: 33 1 30 87 75 40
Fax: 33 1 30 87 79 16

JUNE 2013

Code_Saturne documentation

Code_Saturne version 3.1.0: autovnv tool

contact: saturne-support@edf.fr

TABLE OF CONTENTS

1 Introduction

Autovnv is a small framework to automate the launch of *Code_Saturne* computations and do some operations on new results.

The script needs a directory of previous *Code_Saturne* cases which are candidates to be duplicated. This directory is called **repository**. The duplication is done in a new directory which is called the **destination**.

For each duplicated case, Autovnv is able to compile the user files, to run the case, to compare the obtained checkpoint file with the previous one from the **repository**, and to plot curves in order to illustrate the computations.

For all these steps, Autovnv generate two reports, a global report which summarizes the status of each case, and a detailed report which gives the differences between the new results and the previous ones in the **repository**, and display the defined plots.

In the **repository**, previous results of computations are required only for checkpoint files comparison purpose. They can be also usfull, if the user needs to run specific scripts.

2 Installation and prerequisites

AUTOVNV does not need a specific installation: the related files are installed with the other Python scripts of *Code_Saturne*. Nevertheless, additional prerequisites required are:

- `numpy`,
- `matplotlib`,
- `python-vtk`.

3 Command line options

The command line options can be found with the command: `code_saturne autovnv -h`.

- `-f FILE, --file=FILE`: gives the file of parameters for AUTOVNV. This file is mandatory, and therefore this option must be completed;
- `-q, --quiet`: does not print status messages to stdout;
- `-u, --update`: update installation pathes in scripts (i.e. `SaturneGUI` and `runcase`) only in the repository, reinitialize xml files of parameters and compile;
- `-r, --run`: runs all cases;
- `-c, --compare`: compares chekpoint files between **repository** and **destination**;
- `-p, --post`: postprocess results of computations;
- `-m ADDRESS1 ADDRESS2 ..., --mail=ADDRESS1 ADDRESS2 ...`: addresses for sending the reports.

Examples:

- `code_saturne autovnv -f sample.xml`: duplicates all cases from the **repository** in the **destination**, compile all user files and exits;
- `code_saturne autovnv -f sample.xml -r`: as above, and run all cases if defined in `sample.xml`;
- `code_saturne autovnv -f sample.xml -r -c`: as above, and compares all new checkpoint files with those from the **repository** if defined in `sample.xml`;
- `code_saturne autovnv -f sample.xml -rcp`: as above, and plots results if defined in `sample.xml`;
- `code_saturne autovnv -f sample.xml -r -c -p -m "dt@moulinsart.be dd@moulinsart.be"`: as above, and send the two reports.
- `code_saturne autovnv -f sample.xml -c -p`: compares and plots results in the **destination** already computed.

Note:

The detailed report is generated only if the options `-c, --compare` or `-p, --post` is present in the command line.

4 File of parameters

The file of parameters is a XML formatted ascii file.

4.1 Begin and end of the file of parameters

This example shows the four mandatory first lines of the file of parameters.

```
<?xml version="1.0"?>
<autovnv>
    <repository>/home/dupond/codesaturne/MyRepository</repository>
    <destination>/home/dupond/codesaturne/MyDestination</destination>
```

The third and fourth lines correspond to the definition of the **repository** and **destination** directories. Inside the markups `<repository>` and `<destination>` the user must inform the related directories. If the **destination** does not exit, the directory is created.

The last line of the file of parameters must be:

```
</autovnv>
```

4.2 Case creation and compilation fo the user files

When AUTOVNV is launched, the file of parameters is parsed in order to known which studies and cases from the **repository** should be duplicated in the **destination**. The selection is done with the markups `<study>` and `<case>` as the following example:

```
<?xml version="1.0"?>
<autovnv>
    <repository>/home/dupond/codesaturne/MyRepository</repository>
    <destination>/home/dupond/codesaturne/MyDestination</destination>

    <study label="MyStudy1" status="on">
        <case label="Grid1" run_id="Grid1" status="on" compute="on" post="off"/>
        <case label="Grid2" run_id="Grid2" status="off" compute="on" post="off"/>
    </study>
    <study label="MyStudy2" status="off">
        <case label="k-eps" status="on" compute="on" post="off"/>
        <case label="Rij-eps" status="on" compute="on" post="off"/>
    </study>
</autovnv>
```

The attributes are:

- `label`: the name of the file of the script;
- `status`: must be equal to `on` or `off`, activate or desactivate the markup;
- `compute`: must be equal to `on` or `off`, activate or desactivate the computation of the case;
- `post`: must be equal to `on` or `off`, activate or desactivate the post-processing of the case;
- `run_id`: label of the directory in which the result is stored. If this attribut is missing or set to `run_id=""`, an automatic value will be proposed by the code.

Only the attributes `label`, `status`, `compute` and `post` are mandatory.

If the directory specified by the attribute `run_id` already exists, the comptutation is not performed again. For the post-processing step, the existing results are taking into accout only if no error file is detected in the directory.

With the attribute `status`, a single case or a complete study can be switched off. In the above example, only the case `Grid1` of the study `MyStudy1` is going to be created.

After the creation of the directories in the **destination**, for each case, all user files are compiled. The AUTOVNV stops if a compilation error occurs: neither computation nor comparison nor plot will be performed, even if they are switched on.

Notes:

- During the duplication, every files are copied, except mesh files, for which a symbolic link is used.
- During the duplication, if a file already exists in the **destination**, this file is not overwritten by AUTOVNV.

4.3 Run cases

??

The computations are activated if the option `-r, --run` is present in the command line.

All cases described in the file of parameters with the attribute `compute="on"` are taken into account.

```
<?xml version="1.0"?>
<autovnv>
    <repository>/home/dupond/codesaturne/MyRepository</repository>
    <destination>/home/dupond/codesaturne/MyDestination</destination>

    <study label="MyStudy1" status="on">
        <case label="Grid1" status="on" compute="on" post="off"/>
        <case label="Grid2" status="on" compute="off" post="off"/>
    </study>
    <study label="MyStudy2" status="on">
        <case label="k-eps" status="on" compute="on" post="off"/>
        <case label="Rij-eps" status="on" compute="on" post="off"/>
    </study>
</autovnv>
```

After the computation, if no error occurs, the attribute `compute` is set to `"off"` in the copy of the file of parameters in the **destination**. It is allow to restart AUTOVNV without re-run successfull previous computations.

Note that it is allowed to run several times the same case in a given study. The case has to be repeated in the file of parameters:

```
<?xml version="1.0"?>
<autovnv>
    <repository>/home/dupond/codesaturne/MyRepository</repository>
    <destination>/home/dupond/codesaturne/MyDestination</destination>

    <study label="MyStudy1" status="on">
        <case label="CASE1" run_id="Grid1" status="on" compute="on" post="on">
            <prepro label="grid.py" args="-m grid1.med -p cas.xml" status="on"/>
        </case>
```

```
        <case label="CASE1" run_id="Grid2" status="on" compute="on" post="on"/>
            <prepro label="grid.py" args="-m grid2.med -p cas.xml" status="on"/>
        </case>
    </study>
</autovnv>
```

If nothing is done, the case is repeated without modifications. In order to modify the setup between two runs of the same case, an external script has to be used for change the related setup (see sections 4.5 and 6).

4.4 Compare checkpoint files

The comparison is activated if the option `-c, --compare` is present in the command line.

In order to compare two checkpoint files, markups `<compare>` have to be added as a child of the considered case. In the following exemple, a checkpoint file comparison is switched on for the case *Grid1* (for all variables, with the default threshold), whereas no comparison is planed for the case *Grid2*. The comparison is done by the external script `cs_io_dump` with the option `--diff`.

```
<study label='MyStudy1' status='on'>
    <case label='Grid1' status='on' compute="on" post="off">
        <compare dest="" repo="" status="on"/>
    </case>
    <case label='Grid2' status='on' compute="off" post="off"/>
</study>
```

The attributes are:

- `repo`: id of the results directory in the **repository** for example `repo="20110704-1116"`, if there is a single results directory in the `RESU` directory of the case, the id can be ommitted: `repo=""`;
- `dest`: id of the results directory in the **destination**:
  - → if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
  - → if AUTOVNV is restarted without the run step (with the command line `code_saturne autovnv -f sample.xml -c` for example), the id of the results directory in the **destination** must be given (for example `dest="20110706-1523"`), but if there is a single results directory in the `RESU` directory of the case, the id can be ommitted: `dest=""`, the id will be completed automatically;
- `args`: additional options for the script `cs_io_dump`
  - ◇ `--section`: name of a particular variable;
  - ◇ `--threshold`: real value above which a difference is considered significant (default: $1e-30$ for all variables);
- `status`: must be equal to `on` or `off`: activate or desactivate the markup.

Only the attributes `repo`, `dest` and `status` are mandatory.

Several comparisons with different options are permitted:

```
<study label='MyStudy1' status='on'>
    <case label='Grid1' status='on' compute="on" post="off">
        <compare dest="" repo="" args="--section Pressure --threshold=1000" status="on"/>
```

```
        <compare dest="" repo="" args="--section VelocityX --threshold=1e-5" status="on"/>
        <compare dest="" repo="" args="--section VelocityY --threshold=1e-3" status="on"/>
    </case>
</study>
```

Comparisons results will be sumarized in a table in the file `report_detailed.pdf` (see 5):

Variable Name	Diff. Max	Diff. Mean	Threshold
VelocityX	0.102701	0.00307058	1.0e-5
VelocityY	0.364351	0.00764912	1.0e-3

4.5 Run external additional preprocessing scripts with options

The markup `<prepro>` has to be added as a child of the condidered case.

```
<study label='STUDY' status='on'>
    <case label='CASE1' status='on' compute="on" post="on">
        <prepro label="mesh_coarse.py" args="-n 1" status="on"/>
    </case>
</study>
```

The attributes are:

- `label`: the name of the file of the considered script;
- `status`: must be equal to `on` or `off`: activate or desactivate the markup;
- `args`: additional options to pass to the script.

Only the attributes `label` and `status` are mandatory.

An addionnal option `"-c"` (or `"--case"`) is given by default with the path of the current case as argument (see exemple in section 6 for decoding options).

Note that all options must be processed by the script itself.

Several calls of the same script or to different scripts are permitted:

```
<study label="STUDY" status="on">
    <case label="CASE1" status="on" compute="on" post="on">
        <prepro label="script_pre1.py" args="-n 1" status="on"/>
        <prepro label="script_pre2.py" args="-n 2" status="on"/>
    </case>
</study>
```

All preprocessing scripts are first searched in the `MESH` directory from the current study in the **repository**. If a script is not found, it is searched in the directories of te current case. The main objectif of running such external scripts is to create or modify meshes or to modify the current setup of the related case (see section **??**).

4.6 Run external additional postprocessing scripts with options for a case

The launch of external scripts is activated if the option `-p, --post` is present in the command line.

The markup `<script>` has to be added as a child of the condidered case.

```
<study label='STUDY' status='on'>
    <case label='CASE1' status='on' compute="on" post="on">
        <script label="script_post.py" args="-n 1" dest="" repo="20110216-2147" status="on"/>
    </case>
</study>
```

The attributes are:

- `label`: the name of the file of the considered script;
- `status`: must be equal to `on` or `off`: activate or desactivate the markup;
- `args`: the arguments to pass to the script;
- `repo` and `dest`: id of the results directory in the **repository** or in the **destination**;
  - → if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
  - → if there is a single results directory in the `RESU` directory (either in the **repository** or in the **destination**) of the case, the id can be ommitted: `repo=""` or `dest=""`, the id will be completed automatically.

  If attributes `repo` and `dest` exist, their associated value will be passed to the script as arguments, with options `"-r"` and `"-d"` respectively.

Only the attributes `label` and `status` are mandatory.

Several calls of the same script or to different scripts are permitted:

```
<study label="STUDY" status="on">
    <case label="CASE1" status="on" compute="on" post="on">
        <script label="script_post.py" args="-n 1" status="on"/>
        <script label="script_post.py" args="-n 2" status="on"/>
        <script label="script_post.py" args="-n 3" status="on"/>
        <script label="another_script.py" status="on"/>
    </case>
</study>
```

All postprocessing scripts must be in the `POST` directory from the current study in the **repository**. The main objectif of running external scripts is to create or modify results in order to plot them.

Example of script, which searches printed informations in the listing, note the function to process the passed command line arguments:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os, sys
import string
from optparse import OptionParser

def process_cmd_line(argv):
    """Processes the passed command line arguments."""
    parser = OptionParser(usage="usage: %prog [options]")

    parser.add_option("-r", "--repo", dest="repo", type="string",
```

```
                    help="Directory of the result in the repository")

    parser.add_option("-d", "--dest", dest="dest", type="string",
                    help="Directory of the result in the destination")

    (options, args) = parser.parse_args(argv)
    return options

def main(options):
    m = os.path.join(options.dest, "listing")
    f = open(m)
    lines = f.readlines()
    f.close()

    g = open(os.path.join(options.dest, "water_level.dat"), "w")
    g.write("# time,   h_sim,   h_th\n")
    for l in lines:
       if l.rfind("time, h_sim, h_th") == 0:
           d = l.split()
           g.write("%s  %s  %s\n" % (d[3], d[4], d[5]))
    g.close()

if __name__ == '__main__':
    options = process_cmd_line(sys.argv[1:])
    main(options)
```

4.7 Run external additional postprocessing scripts with options for a study

The launch of external scripts is activated if the option `-p, --post` is present in the command line.

The purpose of this functionality is to create new data based on several runs of cases, and to plot them (see section 4.8).

The markup `<postpro>` has to be added as a child of the condidered study.

```
<study label='STUDY' status='on'>
    <case label='CASE1' status='on' compute="on" post="on"/>
    <postpro label='Grid2' status="on" arg="-n 100">
        <data file="profile.dat">
            <plot fig="1" xcol="1" ycol="2" legend="Grid level 2" fmt='b-p'/>
            <plot fig="2" xcol="1" ycol="3" legend="Grid level 2" fmt='b-p'/>
        </data>
    </postpro>
</study>
```

The attributes are:

- `label`: the name of the file of the considered script;
- `status`: must be equal to `on` or `off`: activate or desactivate the markup;
- `args`: the additional options to pass to the script;

Only the attributes `label` and `status` are mandatory.

The options given to the script in the command line are:

- `-s` or `--study`: label of the current study;

- `-c` or `--cases`: string which contains the list of the cases
- `-d` or `--directories`: string which contains the list of the directories of results.

Additional options can be pass to the script throught the attributes `args`.

Note that all options must be processed by the script itself.

Several calls of the same script or to different scripts are permitted.

4.8 Post-processing: curves

The post-processing is activated if the option `-p, --post` is present in the command line.

The following example shows the drawing of four curves (or plots, or 2D lines) from two files of data (which have the same name `profile.dat`). There are two subsets of curves (i.e. frames with axis and 2D lines), in a single figure. The figure will be saved on the disk in a **pdf** (default) or **png** format, in the `POST` directory of the related study in the **destination**. Each drawing of a single curve is defined as a markup child of a file of data inside a case. Subsets and figures are defined as markup childs of `<study>`.

```
<study label='Study' status='on'>
    <case label='Grid1' status='on' compute="off" post="on">
        <data file="profile.dat" dest="">
            <plot fig="1" xcol="1" ycol="2" legend="Grid level 1" fmt='r-s'/>
            <plot fig="2" xcol="1" ycol="3" legend="Grid level 1" fmt='r-s'/>
        </data>
    </case>
    <case label='Grid2' status='on' compute="off" post="on">
        <data file="profile.dat" dest="">
            <plot fig="1" xcol="1" ycol="2" legend="Grid level 2" fmt='b-p'/>
            <plot fig="2" xcol="1" ycol="3" legend="Grid level 2" fmt='b-p'/>
        </data>
    </case>
    <subplot id="1" legstatus='on' legpos ='0.95 0.95' ylabel="U ($m/s$)" xlabel="Time ($s$)"/>
    <subplot id="2" legstatus='on' legpos ='0.95 0.95' ylabel="U ($m/s$)" xlabel="Time ($s$)"/>
    <figure name="velocity" idlist="1 2" figsize="(4,5)" format="png"/>
</study>
```

4.8.1 Define curves

The curves of computational data are build from data files. These data must be ordered as column and the files should be in results directory in the `RESU` directory (either in the **repository** or in the **destination**). Commentaries are allowed in the file, the head of every commentary line must start with character `#`.

In the file of parameters, curves are defined with two markups: `<data>` and `<plot>`:

- `<data>`: child of markup `<case>`, defines a file of data;
  - → `file`: name of the file of data
  - → `repo` or `dest`: id of the results directory either in the **repository** or in the **destination**;
    - ⇒ if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
    - ⇒ if there is a single results directory in the `RESU` directory (either in the **repository** or in the **destination**) of the case, the id can be ommitted: `repo=""` or `dest=""`, the id will be completed automatically.

The attribute `file` is mandatory, and either `repo` or `dest` must be present (but not the both) even if it is empty.

- `<plot>`: child of markup `<data>`, defines a single curve; the attributes are:
  - → `fig`: id of the subset of curves (i.e. markup `<subplot>`) where the current curve should be plotted;
  - → `xcol`: number of the column in the file of data for the abscisse;
  - → `ycol`: number of the column in the file of data for the ordinate;
  - → `legend`: add a label to a curve;
  - → `fmt`: format of the line, composed from a symbol, a color and a linestyle, for example `fmt="r--"` for a dashed red line;
  - → `xplus`: real to add to all values of the column `xcol`;
  - → `yplus`: real to add to all values of the column `ycol`;
  - → `xfois`: real to multiply to all values of the column `xcol`;
  - → `yfois`: real to multiply to all values of the column `ycol`;
  - → `xerr`: draw horizontal error bar (see section 4.8.5);
  - → `yerr`: draw vertical error bar (see section 4.8.5);
  - → some standard options of 2D lines can be added, for example `markevery="2"` or `markersize="3.5"`. These options are summarized in the table 2. Note that the options which are string of characters must be overquoted likes this: `color="'g'"`.

Property	**Value Type**
alpha	float (0.0 transparent through 1.0 opaque)
antialiased or aa	`True` or `False`
color or c	any matplotlib color
dash_capstyle	`butt`; `round`; `projecting`
dash_joinstyle	`miter`; `round`; `bevel`
dashes	sequence of on/off ink in points ex: `dashes="(5,3)"`
label	any string, same as legend
linestyle or ls	`-`; `--`; `-.`; `:`; `steps`; ...
linewidth or lw	float value in points
marker	`+`; `,`; `.`; `1`; `2`; `3`; `4`; ...
markeredgecolor or mec	any matplotlib color
markeredgewidth or mew	float value in points
markerfacecolor or mfc	any matplotlib color
markersize or ms	float
markevery	`None`; integer; (startind, stride)
solid_capstyle	`butt`; `round`; `projecting`
solid_joinstyle	`miter`; `round`; `bevel`
zorder	any number

Table 2: Options authorized as attributes of the markup `plot`.

The attributes `fig` and `ycol` are mandatory.

Details on 2D lines properties can be found in the `matplotlib` documentation. For more advanced options see section 4.8.7.

4.8.2 Define subsets of curves

A subset of curves is a frame with two axis, axis labels, legend, title and drawing of curves inside. Such subset is called subplot in the nomenclature of `matplotlib`.

`<subplot>`: child of markup `<study>`, defines a frame with severals curves; the attributes are:

- → `id`: id of the subplot, should be an integer;
- → `legstatus`: if `"on"` display the frame of the legend;
- → `legpos`: sequence of the relative coordinates of the center of the legend, it is possible to draw the legend outside the axis;
- → `title`: set title of the subplot;
- → `xlabel`: set label for the x axis;
- → `ylabel`: set label for the y axis;
- → `xlim`: set range for the x axis;
- → `ylim`: set range for the y axis.

The attributes `fig` and `ycol` are mandatory.

For more advanced options see section 4.8.7.

4.8.3 Define figures

Figure is a compound of subset of curves.

`<figure>`: child of markup `<study>`, defines a pictures with a layout of frames; the attributes are:

- → `name`: name of the file to be written on the disk;
- → `idlist`: list of the subplot to be displayed in the figure;
- → `title`: add a title on the top of the figure;
- → `nbrow`: impose a number of row of the layout of the subplots;
- → `nbcol`: impose a number of column of the layout of the subplots;
- → `format`: format of the file to be written on the disk, `"pdf"` (default) or `"png"` [1];
- → standard options of figure can be added (table 3), for example `figsize="(3,4)"`.

Property	Value Type
figsize	width x height in inches; defaults to (4,4)
dpi	resolution; defaults to 200

Table 3: Options authorized as attributes of the markup `figure`.

Details can be found in the matplotlib documentation. For more advanced options see section 4.8.7.

The attributes `name` and `idlist` are mandatory.

[1] Other format could be choosen (eps, ps, svg,...), but the pdf generation with pdflatex will failed.

4.8.4 Experimental or analytical data

A particular markup is provided for curves of experimental or analytical data: `<measurement>`; the attributes are:

→ `file`: name of the file to be read on the disk;

→ `path`: path of the directory where the file of data is. the path could be ommitted (`path=""`), and in this case, the file will be searched recursively in the directories of the considered study.

The attributes `file` and `path` are mandatory.

In order to draw curves of experimental or analytical data, the markup `<measurement>` should be used with the markup `<plot>` as illustrated below:

```
<study label='MyStudy' status='on'>
    <measurement file='exp1.dat' path=''>
            <plot fig='1' xcol='1' ycol='2' legend='U Experimental data'/>
            <plot fig='2' xcol='3' ycol='4' legend='V Experimental data'/>
    </measurement>
    <measurement file='exp2.dat' path =''>
            <plot fig='1' xcol='1' ycol='2' legend='U Experimental data'/>
            <plot fig='2' xcol='1' ycol='3' legend='V Experimental data'/>
    </measurement>
    <case label='Grid1' status='on' compute="off" post="on">
        <data file="profile.dat" dest="">
            <plot fig="1" xcol="1" ycol="2" legend="U computed" fmt='r-s'/>
            <plot fig="2" xcol="1" ycol="3" legend="V computed" fmt='b-s'/>
        </data>
    </case>
</study>
<subplot id="1" legstatus='on'  ylabel="U ($m/s$)" xlabel= "$r$ ($m$)" legpos ='0.05 0.1'/>
<subplot id="2" legstatus='off' ylabel="V ($m/s$)" xlabel= "$r$ ($m$)"/>
<figure name="MyFigure" idlist="1 2"  figsize="(4,4)" />
```

4.8.5 Curves with error bar

In order to draw horizontal and vertical error bar, it is possible to specify to the markup `<plot>` the attributes `xerr` and `yerr` respectively. The value of theses attributes could be:

- a single column number where the average uncertainty is in the file of data:

  ```
  <measurement file='axis.dat' path =''>
      <plot fig='1' xcol='1' ycol='3' legend='Experimental data' xerr='2' />
  </measurement>
  ```

- a sequence of two column numbers where the minimum and the maximum of the uncertainty are in the file of data:

  ```
  <data file='profile.dat' dest="">
      <plot fig='1' xcol='1' ycol='2' legend='computation' yerr='3 4' />
  </data>
  ```

Notes:

The attributes `xerr` and `yerr` could not be both at the same time in the markup `plot`. If horizontal and vertical bars has be drawn together, repeat the markup `plot` like this:

```
<measurement file='axis.dat' path =''>
    <plot fig='1' xcol='1' ycol='3' legend='Experimental data' xerr='2' />
    <plot fig='1' xcol='1' ycol='3' yerr='4' />
</measurement>
```

4.8.6 Monitoring points or probes

A particular markup is provided for curves of probes data: `<probes>`; the attributes are:

- `file`: name of the file to be read on the disk;
- `fig`: id of the subset of curves (i.e. markup `<subplot>`) where the current curve should be plotted;
- `dest`: id of the results directory in the **destination**:
  - → if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
  - → if AUTOVNV is restarted without the run step (with the command line `code_saturne autovnv -f sample.xml -c` for example), the id of the results directory in the **destination** must be given (for example `dest="20110706-1523"`), but if there is a single results directory in the `RESU` directory of the case, the id can be ommitted: `dest=""`, the id will be completed automatically;

The attributes `file`, `fig` and `dest` are mandatory.

In order to draw curves of probes data, the markup `<probes>` should be used as a child of a markup `<case>` as illustrated below:

```
<study label='MyStudy' status='on'>
    <measurement file='exp1.dat' path=''>
        <plot fig='1' xcol='1' ycol='2' legend='U Experimental data'/>
    </measurement>
    <case label='Grid1' status='on' compute="off" post="on">
        <probes file="probes_U.dat" fig ="2" dest="">
        <data file="profile.dat" dest="">
            <plot fig="1" xcol="1" ycol="2" legend="U computed" fmt='r-s'/>
        </data>
    </case>
</study>
<subplot id="1" legstatus='on'  ylabel="U ($m/s$)" xlabel= "$r$ ($m$)" legpos ='0.05 0.1'/>
<subplot id="2" legstatus='on'  ylabel="U ($m/s$)" xlabel= "$time$ ($s$)" legpos ='0.05 0.1'/>
<figure title="Results" name="MyFigure" idlist="1"/>
<figure title="Grid1: probes for velocity"  name="MyProbes" idlist="2"/>
```

4.8.7 Matplotlib raw commands

The file of parameters allows to execute additional matplotlib commands (i.e Python commands), for curves (2D lines), or subplot, or figure. For every object drawn, `Autovnv` associate a name to this object that can be reused in standard matplotlib commands. Therefore, childs markup `<plt_command>` could be added to `<plot>`, `<subplot>` or `<figure>`.

It is possible to add commands with **Matlab style** or **Python style**. For the Matlab style, commands are called as methods of the module `plt`, and for Python style commands or called as methods of the instance of the graphical object.

Matlab style and Python style commands can be mixed.

- curves or 2D lines: when a curve is drawn, the associated name are `line` and `lines` (with `line = lines[0]`).

```
<plot fig="1" xcol="1" ycol="2" fmt='g^' legend="Simulated water level">
    <plt_command>plt.setp(line, color="blue")</plt_command>
    <plt_command>line.set_alpha(0.5)</plt_command>
</plot>
```

- subset of curves (subplot): for each subset, the associated name is `ax`:

```
<subplot id="1" legend='Yes' legpos ='0.2 0.95'>
    <plt_command>plt.grid(True)</plt_command>
    <plt_command>plt.xlim(0, 20)</plt_command>
    <plt_command>ax.set_ylim(1, 3)</plt_command>
    <plt_command>plt.xlabel(r"Time ($s$)", fontsize=8)</plt_command>
    <plt_command>ax.set_ylabel(r"Level ($m$)", fontsize=8)</plt_command>
    <plt_command>for l in ax.xaxis.get_ticklabels(): l.set_fontsize(8)</plt_command>
    <plt_command>for l in ax.yaxis.get_ticklabels(): l.set_fontsize(8)</plt_command>
    <plt_command>plt.axis([-0.05, 1.6, 0.0, 0.15])</plt_command>
    <plt_command>plt.xticks([-3, -2, -1, 0, 1])</plt_command>
</subplot>
```

4.9 Post-processing: scalar map

The post-processing is activated if the option `-p, --post` is present in the command line.

AUTOVNV is able to draw colored maps from results on cut planes of the computational domain. The results must be a set of ensight files. The following example shows the drawing of a scalar map using the master file of result `RESULTS.case`. The figure will be saved on the disk in a **png** format, in the `POST` directory of the related study in the **destination**. Each drawing of a single scalar map is defined as a markup child of a file of data inside a case.

```
<case label='Grid1' status='on' compute="on" post="on">
    <resu file="RESULTS.case" dest="">
        <scalar name="alpha1" variable="alpha1" normal="(0,0,1)">
            <scale legend="Water fraction" levels="10" />
            <title label="Grid level 1" fontsize="15"/>
            <axes/>
        </scalar>
        <scalar name="pressure" variable="Pressure"/>
    </resu>
</case>
```

In the file of parameters, scalar map are defined with two markups: `<resu>` and `<scalar>`. Three other markups can be used: `<scale>`, `<title>` and `<axes>` as childs of the markups `<scalar>`.

- `<resu>`: child of markup `<case>`, defines the master file of set of ensight files;
  - → `file`: name of the master ensight file
  - → `repo` or `dest`: id of the results directory either in the **repository** or in the **destination**;
    - ⇒ if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);
    - ⇒ if there is a single results directory in the `RESU` directory (either in the **repository** or in the **destination**) of the case, the id can be ommitted: `repo=""` or `dest=""`, the id will be completed automatically.

The attribute `file` is mandatory, and either `repo` or `dest` must be present (but not the both) even if it is empty.

- `<scalar>`: child of markup `<resu>`, defines a scalar map;
  - → `name`: name of the image file on the disk
  - → `variable`: name of the variable to draw (must be present in the master ensight file)
  - → `normal`: tuple to set the normal of the cut plane of the computational domain
  - → `center`: tuple to set the center of the cut plane of the computational domain
  - → `stretch`: tuple to stretch the view of the cut plane
  - → `time-step`: select the physical time of the value to display
  - → `size`: size of the png image
  - → `zoom`: adjust the zoom for the point of view of the cut plane
  - → `wireframe`: switch on or off the mapping on the grid[2]

  See default values for these attributes in the table 4.

Attributes	**Default values**
normal	(0.,0.,1.)
center	(0.,0.,0.)
stretch	(1.,1.,1.)
time-step	-1 *i.e. the last record*
size	(500,400)
zoom	1.0
wireframe	`off`

Table 4: Default attributes values of the markup `scalar`.

The attributes `name` and `variable` are mandatory.

- `<scale>`: child of markup `<scalar>`, defines a color bar;
  - → `color`: set a predefined palette in "hsv", "gray", "hot", "flag", "jet", "blue_to_yellow", "spring", "summer", "winter", "autumn"
  - → `range`: tuple to set the range of values to display
  - → `coord`: tuple to set the normalized coordinates of the color bar
  - → `levels`: number of levels for the color bar
  - → `height`: set the height of the color bar
  - → `width`: set the width of the color bar
  - → `position`: simple predefined position for the color bar in "North", "South", "West", or "East"
  - → `legend`: legend of the color bar
  - → `fontsize`: font size for the legend
  - → `format`: float format to display the value of the levels of the color bar

  See default values for these attributes in the table 5.

  No attribute is mandatory.

- `<title>`: child of markup `<scalar>`, defines a title for the scalarmap;

[2]But the view might not be the true computational grid.

Attributes	Default values
color	*default vtk palette*
range	*all values*
coord	(0.9,0.09)
levels	10
height	0.8
width	0.1
position	East
legend	*no legend*
fontsize	20
format	%4.4f

Table 5: Default attributes values of the markup `scale`.

→ `label`: text for the title

→ `fontsize`: font size for the title

→ `coord`: tuple to set the normalized coordinates of the title

See default values for these attributes in the table 6.

Attributes	Default values
label	*no title*
fontsize	20
coord	(0.5,0.99)

Table 6: Default attributes values of the markup `title`.

No attribute is mandatory.

- `<axes>`: child of markup `<scalar>`, defines axes bars around the scalarmap;

  → `fontsize`: font size for the ticks of the axes

  → `format`: float format to display the ticks of the axes

  → `levels`: number of ticks of all axes

  → `xlabel`: text of the X axe

  → `ylabel`: text of the Y axe

  → `zlabel`: text of the Z axe

  See default values for these attributes in the table 7.

Attributes	Default values
fontsize	20
format	%6.3g
levels	3
xlabel	*no text*
ylabel	*no text*
zlabel	*no text*

Table 7: Default attributes values of the markup `axes`.

No attribute is mandatory.

4.9.1 VTK raw commands

The file of parameters allows to execute additional vtk commands (i.e Python commands). This possibility is mainly useful for interaction with actors and the camera. For each actor drawn, `Autovnv` associate a name to this object that can be reused in standard vtk commands. Therefore, childs markup `<vtk_command>` could be added to `<scalar>`.

The defined actors and camera are:

→ `grid`: the scalar map,

→ `axes` (if `<axes>` exits),

→ `legend` (if `<scale>` exits),

→ `title` (if `<title>` exits),

→ `cam`: the parameters of the camera could be very important, associated to the zoom to display correctly the scalar map.

```
<resu file="RESULTS.case" dest="">
    <scalar name="alpha1_case1" variable="alpha1" normal="(0,1,0)" zoom="0.3">
        <scale legend="Water fraction" levels="5" fontsize="20"/>
        <title label="Grid level 1"/>
        <axes/>
        <vtk_command>cam.SetViewUp(1, 0, 1)</vtk_command>
        <vtk_command>axes.ZAxisVisibilityOff()</vtk_command>
        <vtk_command>legend.GetTitleTextProperty().BoldOn()</vtk_command>
        <vtk_command>legend.SetOrientationToHorizontal()</vtk_command>
    </scalar>
</resu>
```

4.10 Post-processing: input files

The post-processing is activated if the option `-p, --post` is present in the command line.

AUTOVNV is able to include files into the final detailed report. These files must be in the directory of results either in the **destination** or in the **repository**. The following example shows the inclusion of three files: `performance.log` and `setup.log` from the **destination**, and a `performance.log` from the **repository**:

```
<case label='Grid1' status='on' compute="on" post="on">
    <input dest="" file="performance.log"/>
    <input dest="" file="setup.log"/>
    <input repo="" file="performance.log"/>
</case>
```

In the file of parameters, input files are defined with markups `<input>` as childs of a single markup `<case>`. The attributes of `<input>` are:

→ `file`: name of the file to be included

→ `repo` or `dest`: id of the results directory either in the **repository** or in the **destination**;

⇒ if the id is not known already because the case has not yet run, just let the attribute empty `dest=""`, the value will be updated after the run step in the **destination** directory (see section 5);

⇒ if there is a single results directory in the `RESU` directory (either in the **repository** or in the **destination**) of the case, the id can be ommitted: `repo=""` or `dest=""`, the id will be completed automatically.

The attribute `file` is mandatory, and either `repo` or `dest` must be present (but not the both) even if it is empty.

5 Output and restart

AUTOVNV produces several files in the **destination** directory:

- `report.txt`: standard output of the script;
- `auto_vnv.log`: log of the code and the `pdflatex` compilation;
- `report_global.pdf`: summary of the compilation, run, comparison, and plot steps;
- `report_detailed.pdf`: details the comparison and display the plot;
- `sample.xml`: udpated file of parameters, useful for restart the script if an error occurs.

After the computation of a case, if no error occurs, the attribute `compute` is set to `"off"` in the copy of the file of parameters in the **destination**. It is allow a restart of AUTOVNV without re-run successfull previous computations. In the same manner, all empty attributes `repo=""` and `dest=""` are completed in the udpated file of parameters.

6 Tricks

- How to comment markups in the file of parameter ?

  The opening and closing signs for commantaries are `<!--` and `-->`. In the following example, nothing from the study `MyStudy2` will be read:

```
<?xml version="1.0"?>
<autovnv>
    <repository>/home/dupond/codesaturne/MyRepository</repository>
    <destination>/home/dupond/codesaturne/MyDestination</destination>

    <study label="MyStudy1" status="on">
        <case label="Grid1" status="on" compute="on" post="on"/>
        <case label="Grid2" status="on" compute="off" post="on"/>
    </study>
    <!--
    <study label="MyStudy2" status="on">
        <case label="k-eps" status="on" compute="on" post="on"/>
        <case label="Rij-eps" status="on" compute="on" post="on"/>
    </study>
    -->
</autovnv>
```

- How to add text in a figure ?

  It is possible to use raw commands:

```
<subplot id='301' ylabel ='Location ($m$)' title='Before jet -0.885' legstatus='off'>
    <plt_command>plt.text(-4.2, 0.113, 'jet')</plt_command>
    <plt_command>plt.text(-4.6, 0.11, r'$\downarrow$', fontsize=15)</plt_command>
</subplot>
```

- Adjust margins for layout of subplots in a figure.

  You have to use the raw command `subplots_adjust`:

```
<subplot id="1" legend='Yes' legpos ='0.2 0.95'>
    <plt_command>plt.subplots_adjust(hspace=0.4, wspace=0.4, right=0.9,
                left=0.15, bottom=0.2, top=0.9)</plt_command>
</subplot>
```

- How to find a syntax error in the XML file ?

  When there is a misprint in the file of parameters, AUTOVNV indicates the location of the error with the line and the column of the file:

```
my_case.xml file reading error.

This file is not in accordance with XML specifications.

The parsing syntax error is:

my_case.xml:86:12: not well-formed (invalid token)
```

- How to set a logarithmic scale ?

  The following raw commands have to be used:

```
<subplot id="2" title="Grid convergence" xlabel="Number of cells" ylabel="Error (\%)">
    <plt_command>ax.set_xscale('log')</plt_command>
    <plt_command>ax.set_yscale('log')</plt_command>
</subplot>
```

- How to create a mesh automatically with SALOME ?

  The flollowing example shows how to create a mesh with a SALOME command file:

```
<study label="STUDY" status="on">
    <case label="CASE1" status="on" compute="on" post="on">
        <prepro label="salome.sh" args="-t -u my_mesh.py" status="on"/>
    </case>
</study>
```

  with the script `salome.sh` (depending of the local installation of SALOME):

```
#!/bin/bash

export ROOT_SALOME=/home/salome/salome-640/Salome-V6_4_0-c7-v2
source /home/salome/salome-640/Salome-V6_4_0-c7-v2/salome_prerequisites_V6_4_0_appli.sh
source /home/salome/salome-640/Salome-V6_4_0-c7-v2/salome_modules_V6_4_0.sh

/home/salome/salome-640/appli_V6_4_0/bin/salome/runSalome $*
```

  and the script of SALOME commands `my_mesh.py`:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import geompy
import smesh

# create a box
box = geompy.MakeBox(0., 0., 0., 100., 200., 300.)
idbox = geompy.addToStudy(box, "box")

# create a mesh
tetra = smesh.Mesh(box, "MeshBox")
```

```
algo1D = tetra.Segment()
algo1D.NumberOfSegments(7)

algo2D = tetra.Triangle()
algo2D.MaxElementArea(800.)

algo3D = tetra.Tetrahedron(smesh.NETGEN)
algo3D.MaxElementVolume(900.)

# compute the mesh
tetra.Compute()

# export the mesh in a MED file
tetra.ExportMED("./my_mesh.med")
```

- How to make a grid convergence study ?

  The following exemple show how to run twice the same case, with a new mesh:

```
<case compute="on" label="CANON" run_id="50CELLS" post="on" status="on">
    <prepro label="grid.py" args="-m canon50.des.gz -p supercanon.xml" status="on"/>
        <data dest="" file="probes_Pressure.dat">
        <plot fig="1" fmt="r-s" legend="Grid level 1" xcol="1" ycol="2"/>
        <plot fig="2" fmt="r-s" xcol="1" ycol="3"/>
    </data>
</case>

<case compute="on" label="CANON" run_id="500CELLS" post="on" status="on">
    <prepro label="grid.py" args="-m canon500.des.gz -p supercanon.xml" status="on"/>
    <data dest="" file="probes_Pressure.dat">
        <plot fig="1" fmt="b-d" legend="Grid level 2" xcol="1" ycol="2"/>
        <plot fig="2" fmt="b-d" xcol="1" ycol="3"/>
    </data>
</case>
```

  In order to change the name of the mesh inside the file of parameters of the code a script `grid.py` has to be written in the directory `MESH` of the study or in the directory `DATA` of the related case:

```
#!/usr/bin/env python
import os, sys
import string
from optparse import OptionParser
sys.path.append('/home/dupond/code/lib/python2.7/site-packages/code_saturne')
sys.path.append('/home/dupond/code/lib/python2.7/site-packages/code_saturne/Base')
sys.path.append('/home/dupond/code/python2.7/site-packages/code_saturne/Pages')

def process_cmd_line(argv):
    """Processes the passed command line arguments."""
    parser = OptionParser(usage="usage: %prog [options]")

    parser.add_option("-c", "--case", dest="case", type="string",
                      help="Directory of the current case")

    parser.add_option("-p", "--param", dest="param", type="string",
                      help="Name f the file of parameters")

    parser.add_option("-m", "--mesh", dest="mesh", type="string",
                      help="Name of the new mesh")

    (options, args) = parser.parse_args(argv)
```

```
    return options

def main(options):
    from cs_package import package
    from Base.XMLengine import Case
    from Base.XMLinitialize import XMLinit
    from Pages.SolutionDomainModel import SolutionDomainModel

    fp = os.path.join(options.case, "DATA", options.param)
    if os.path.isfile(fp):
        try:
            case = Case(package = package(), file_name = fp)
        except:
            print("Parameters file reading error.\n")
            print("This file is not in accordance with XML specifications.")
            sys.exit(1)

        case['xmlfile'] = fp
        case.xmlCleanAllBlank(case.xmlRootNode())
        XMLinit(case).initialize()
        s = SolutionDomainModel(case)
        l = s.getMeshList()
        s.delMesh(l[0])
        s.addMesh((options.mesh, None))
        case.xmlSaveDocument()

if __name__ == '__main__':
    options = process_cmd_line(sys.argv[1:])
    main(options)
```