

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

MARCH 2012

*Code\_Saturne* documentation

***Code\_Saturne* version 2.2.0: autovnv tool**

contact: [saturne-support@edf.fr](mailto:saturne-support@edf.fr)



|         |   |  |
|---------|---|--|
| EDF R&D | <i>Code_Saturne</i> version 2.2.0: autovnv tool | <i>Code_Saturne</i><br>documentation<br>Page 1/ <a href="#">14</a> |
|---------|---|--|

## TABLE OF CONTENTS

|          |   |    |
|----------|---|----|
| <b>1</b> | <b>Introduction</b>   | 2  |
| <b>2</b> | <b>Installation and prerequisites</b>                         | 2  |
| <b>3</b> | <b>Command line options</b>                                   | 3  |
| <b>4</b> | <b>File of parameters</b>                                     | 3  |
| 4.1      | BEGIN AND END OF THE FILE OF PARAMETERS                       | 3  |
| 4.2      | CASE CREATION AND COMPILATION OF THE USER FILES               | 4  |
| 4.3      | RUN CASES   | 5  |
| 4.4      | COMPARE CHECKPOINT FILES                                      | 5  |
| 4.5      | RUN EXTERNAL ADDITIONAL PREPROCESSING SCRIPTS WITH ARGUMENTS  | 6  |
| 4.6      | RUN EXTERNAL ADDITIONAL POSTPROCESSING SCRIPTS WITH ARGUMENTS | 7  |
| 4.7      | POST-PROCESSING   | 9  |
| 4.7.1    | Define curves   | 9  |
| 4.7.2    | Define subsets of curves                                      | 10 |
| 4.7.3    | Define figures  | 11 |
| 4.7.4    | Experimental data   | 12 |
| 4.7.5    | Curves with error bar   | 12 |
| 4.7.6    | Matplotlib raw commands                                       | 13 |
| <b>5</b> | <b>Output and restart</b>                                     | 13 |
| <b>6</b> | <b>Tricks</b>   | 14 |

## 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*. Additional prerequisites required are:

- numpy,
- matplotlib.

### 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;
- `-r`, `--run`: runs all cases;
- `-c`, `--compare`: compares checkpoint 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 -r -c -p`: 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 exist, the directory is created.

The last line of the file of parameters must be:

```
</autovnv>
```

## 4.2 Case creation and compilation for the user files

When AUTOVNV is launched, the file of parameters is parsed in order to know 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" status="on" compute="on" post="off"/>
    <case label="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 deactivate the markup;
- **compute**: must be equal to **on** or **off**, activate or deactivate the computation of the case;
- **post**: must be equal to **on** or **off**, activate or deactivate the post-processing of the case.

All these attributes are mandatory.

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.

### 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 condidered 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 deactivate 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 summarized 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 arguments

The markup `<prepro>` has to be added as a child of the considered 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 deactivate the markup;
- **args**: the command line to pass to the script.

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">
    <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 must be in the MESH directory from the current study in the **repository**. The main objectif of running such external scripts is to create or modify meshes.

The following 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")
```

## 4.6 Run external additional postprocessing scripts with arguments

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 considered 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 deactivate 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 omitted: **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 Post-processing

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 **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)" />
</study>

```

### 4.7.1 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 omitted: `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 4.7.5);
  - `yerr`: draw vertical error bar (see section 4.7.5);
  - 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'"`.

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 4.7.6.

## 4.7.2 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 several 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;

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

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

- **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 [4.7.6](#).

### 4.7.3 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;
- standard options of figure can be added (table [3](#)), for example **figsize="(3,4)"**.

Details can be found in the matplotlib documentation. For more advanced options see section [4.7.6](#).

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

| 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`.

## 4.7.4 Experimental data

A particular markup is provided for curves of experimental 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 experimental data is. the path could be omitted (`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 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)" />
```

## 4.7.5 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>
```

### 4.7.6 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.

- 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: 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>
  <plt_command>plt.subplots_adjust(hspace=0.4, wspace=0.4, right=0.9, left=0.15)</plt_command>
</subplot>
```

## 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`: updated 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 updated file of parameters.

## 6 Tricks

- How to comment markups in the file of parameter ?

The opening and closing signs for commentaries 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>
```