ENFLO SOFTWARE MATLAB SUITE V1.7

Software and example guide v 1.1 – 3 Sept 2019

This guide introduces the main functionalities of the software, also by means of examples which make use of most of the features

Dr. Davide Marucci
davide.marucci23@gmail.com

Contents

EnFlo Software MATLAB Suite - Overview

The EnFlo Software MATLAB Suite contains different codes which allow to

- Plot profiles in multiple graphs
- Fit the profiles with fitting curves defined by the user, show the resultant profiles and output the coefficients in a file
- Plot contour and scatter graphs, streamlines and vectors on planes with any possible orientation in the space
- Insert the 2D contour plots in a 3D geometry or cut the geometry on the plane identified by the measuring points and show everything in a 2D graph
- Show the measuring points in a 3D geometry (either in combination with the previous graphs or as a standalone)
- Post-process data sampled with the seven-hole probe to obtain the three components of velocity.

All the software is written with MATLAB 2019a and compiled in an executable file which needs MATLAB runtime 9.6 to run. The EXE accepts as inputs tab delimited input files and STL geometry files. The communication and synchronization with other programmes take place with the TCP/IP standard, on a port specified in the file EnFloSoftwareMatlabSuiteSettings.xls.

Once started the EnFloSoftwareMatlabSuite.exe will act as a server and wait for the client to connect on the defined port and send its job request. Four inputs are possible:

- 01; (followed by a series of empty spaces to fill the string length specified in EnFloSoftwareMatlabSuiteSettings.xls) will call the 1Dplotter and produce profile plots
- 02; (followed by a series of empty spaces to fill the string length specified in EnFloSoftwareMatlabSuiteSettings.xls) will call the 2Dplotter and produce 2D/3D plots
- 03; (followed by a series of empty spaces to fill the string length specified in EnFloSoftwareMatlabSuiteSettings.xls) will call the LPplotter and will plot the measuring point locations in the space
- 04; (followed by the calibration file name ending with a ";" and then a series of empty spaces to fill the string length specified in EnFloSoftwareMatlabSuiteSettings.xls) will call the 7HP code to post-process seven-hole probe data

Once the input string has been received, the software loads the input, data and geometry files that must be previously placed in the appropriated folders. The input file is a tab delimited file which contains all the settings needed by the software to output the desired graphs. The first row indicates the variable name. All the variable names must be present in the input files, otherwise an error string will be issued. The software will compute the instructions contained in the input file and provide as output the figures in the desired format. Once the job is finished, the software will send a string message on the TCP port with the same length as the input. It will have the same first three characters of the input. If no error or warnings have been produced, the following characters will be empty spaces, otherwise the text of the error/warnings will be added. At the same time the error/warning text will be written in a log file (ErrorLog.txt). The software will then wait for another client to connect with a new job.

The software will stop only if either a not codified error occurs (a window with a description of the error will appear in this case), or the process is killed from the task manager (or equivalent).

Input files

The input file is a tab-separated file used to set the variables, which 1Dplotter, 2Dplotter and LPplotter need to run. The variables are divided in categories depending on their function and labelled with the first letter of their name. As example the variable *fAxisLabelY* starts with the letter "f" which stays for "figure", meaning that the variable will modify a property of the figure. These are all the variable categories (not all of them are available in all the three plotters of the software)

- **f**: figure. Variables which modify the figure properties
- **p**: profile. Only in 1Dplotter, variables which refer to the single profile in the figure
- **l**: location. Variables which refer to the location plotter (used to plot the measuring point locations in the 3D geometry). Note: this category is not present in the LPplotter
- **c**: contour. Variables to modify the contour graph properties (only in 2Dplotter)
- **t**: streamline. Variables to modify the plot of streamlines (only in 2Dplotter)
- **v**: vectors. Variables to modify the plot of vector fields (only in 2Dplotter)
- **d**: 3D. Variables to modify the plotting of 3D graphs (only in 2Dplotter)
- **s**: single plot. Variables to modify the saving of the figures in the so-called singlePlotter (one figure per file).
- **m**: multi plot. Variables to modify the saving of the figures in the so-called multiPlotter (one file for all the figures arranged in a grid).

Some variables can be labelled by more categories, in this case their name will start with more labelling letters (e.g. gldFaceColor refers to the categories g, l and d).

In each input file the first row contains all the variable names. The order of the columns does not matter, except for the first column, for which no empty cells are allowed). The management of the rows depends on the considered plotter.

Input files – 1Dplotter

In the 1Dplotter each row refers to one profile. Let us consider for example a job constituted by 3 figures in which the first has got 2 profiles, the second 1 and the third 3. In this case the first two rows of the input file (after the header with the variable names) will control the first and the second profile of the first figure, the third row will control the only profile of the second figure, the rows from the forth to the sixth will control the three profiles of the last figure. Having said that, only the variables controlling the single profiles (so labelled with "p") must be specified for each row, the others can be specified only in the row controlling the first profile of each figure (but if indicated also elsewhere this will not cause problems, simply the information in the cell will not be used). A further exception is the variables labelled with "m" which must be indicated only in the first row (again after the headers, so it is technically the second row of the file).

List of all input variables with explanation - 1Dplotter

The variables indicated with [] allow single numbers, the ones indicated with {} are arrays (containing string of texts or multiple numbers separated by spaces or ;)

```
%% Figure settings (read only in the first row of each new figure in the
input.xls file)
pFigureNr = []; %Figure nr. to be specified for each row as first column in
Input1D.xls
fAxisLabelX = {};
fAxisLabelY = {};
fAxisLabelYrightZloc = {}; %Label for either the Y-right axis (in figure
with double Y axis) or the Z axis (in location plotter)
fTitle = {};
fAxisLimX = {}; %[Min;Max] works only if both Min and Max are specified (if
"-inf" or "inf" is set the limit will be the smallest or largest point)
fAxisLimY = {}; %[Min;Max] works only if both Min and Max are specified (if
"-inf" or "inf" is set the limit will be the smallest or largest point)
fAxisLimYrightZloc = {}; %[Min;Max] Axis limits for either the Y-right axis
(in figure with double Y axis) or the Z axis (in location plotter)
fLogAxis = {}; %[x-axis;yleft-axis;yright-axis] logarithmic axes. 0:
linear, 1: log
fInterpreterText = {}; % ['latex','none']
fShowFigure = []; %Show figure in interactive window [0,1]
fFigRenderer = {}; % ['opengl','painters']
fGrid = {}; %[x-axis;y-axis;z-axis] Show axis grid in figure (0,1). N.B. z-
axis setting works only in Location plotter
fGridMinor = {}; %[x-axis;y-axis;z-axis] Show axis minor grid in figure
(0,1). N.B. z-axis setting works only in Location plotter
fTickLabel = {}; %[x-axis;yleft-axis;yright/Zloc-axis] Show axis tick
labels (0,1)
fFontSize = [];
fShowLegend = []; %[0,1]
fShowLegendBox = []; %[0,1]
fLegendLocation =
{}; %['best','north','south','bestoutside','northeastoutside',etc.]
fLegendTitle = {}; %Title to add above the legend (if empty no legend title
is added)
fErrorbarOrientation = {}; %['none','horizontal','vertical','both']
fErrorbarColumn = []; %[1, 2] 1 for same lower and upper limit in one
column, 2 for lower and upper limit specified in different columns
fYrightAxisColor = {}; %Set color of right Y axis (if any)

%% Profile plot settings (read in each row of the input.xls file)
pAxisYright = []; %[0,1] True if the profile refers to a right Y axis,
otherwise false
pLegendLabel = {};
pLineColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal (e.g.
#FF0000)
pLineStyle = {}; %['-','--',':','-.','none']
pLineWidth = [];
pLineMarker =
{}; %['o','+','*','.','x','s','d','^','v','>','<','p','h','none']
pMarkerSize = [];
pFitOn = []; %[0,1] Activate fitting
pFitIndepAxis = {}; %['X','Y']
pFitEquation = {}; %'0.06*log((Y - C(1))/C(2))'
pFitStart = {}; % List of starting points (write "[" and "]" at beginning
and end). One or more values (separated by space)
% for each fitting coefficient (separated by ;). If more than one starting
point for each coeff
```

```
% is indicated, the fitting will be performed for each starting points set
and the one producing
% the smallest fitting error will be selected
pFitLowerBound = {}; % List of lower bounds for coeff research. One for
each fitting coefficient (separated by ;)
pFitUpperBound = {}; % List of upper bounds for coeff research. One for
each fitting coefficient (separated by ;)
pFitIndepAxisFitRange = {}; %Min and Max independent axis values inside
which performing the fitting (separated by ;)
pFitIndepAxisPlotRange = {}; %Min and Max independent axis values inside
which plotting the fitting curve (separated by ;)

%% Singleplotter settings (read only in the first row of each new figure in
the input.xls file)
sSavePlot = []; % Save single plot [0,1]
sFormatFig = {}; % ['fig','png','jpeg','bmp','pdf','meta',etc.]
sLeftEdge = []; %in cm (This value is ignored when saving a figure to a
nonpage format, such as a PNG or EPS format.)
sBottomEdge = []; %in cm (This value is ignored when saving a figure to a
nonpage format, such as a PNG or EPS format.)
sPlotHeight = []; %in cm
sPlotWidth = []; %in cm

%% Location plot settings
lLocationAct = []; %Activate locationPlotter [0,1]
lListFiguresShown = {}; %(FigureNr1;FigureNr2;...)List of figure nr. whose
points are shown in the location plotter
lLocationFilename = {}; %e.g. 'Geometry1.stl'. N.B. Geometry must be in STL
format
lLocationTranslations = {};% (X;Y;Z) Translation to apply to the stl
geometry to align its origin with the plot data origin
lLocationRotations = {};% (angleX;angleY;angleZ) angles (in degrees) to
rotate to the stl geometry to align its axes with the plot data axes. The
rotation will be done in the order around axis Z, Y and as last X
lFaceColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
lEdgeColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
lFaceTransparency = []; %Face surface transparency value from 0 to 1 (where
0 is invisible and 1 solid)
lEdgeTransparency = []; %Edge transparency value from 0 to 1 (where 0 is
invisible and 1 solid)
lViewAngles = {}; %(az;el) Set the viewing angles for the figure in degrees
(azimuth and elevation angles)
lRefPointCoord = {}; %(x;y;z) Coordinates of reference point (e.g. source
location) (leave blank if not needed)
lRefPointColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
(leave blank if not needed)
lRefPointSize = []; % e.g. 20 (leave blank if not needed)
lRefArrowBaseCoord = {}; %(x;y;z) Coordinates of reference arrow base point
(e.g. wind direction) (leave blank if not needed)
lRefArrowComponents = {}; %(x;y;z) Lengths of reference arrow in each
direction (leave blank if not needed)
lRefArrowColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
(leave blank if not needed)
lRefArrowSize = []; % (e.g. 2) Specify the arrow line width and head size
(they have to be the same) (leave blank if not needed)

%% Multiplotter settings (read only in the first row of the input.xls file)
mPlotAct = []; % Save all plots in one file [0,1]
mFormatFig = {}; %['pdf','emf','png']
mResolution = {}; %works only if Renderer is opengl
mMainTitle = {};
```

```
mGraphsxColumn = [];
mGraphsxRow = [];
mLeftEdge = [];
mRightEdge = [];
mTopEdge = [];
mBottomEdge = [];
mSpaceX = [];
mSpaceY = [];
mPlotHeight = [];
mPlotWidth = [];
mMainTitleFontSize = [];
mFontSize = [];
mShowLegendBox = [];
mLegendTitle = {};
mInterpreterText = {};  % ['latex','none'] used only for multiplot legend
and main title
mFigRenderer = {}; % ['opengl','painters']
mLegendLocation = {}; %[X,Y] location of the centre of the legend (where 0
is the right paper edge, 1 the left edge)
mLegendColumnNr = []; %Number of columns in which the legend is arranged
```

Input files – 2Dplotter

In the 2Dplotter each row of the input file refers to the figure number specified in the first column (fFigureNr). This means that multiple rows may refer to the same figure if fFigureNr is the same. In this way multiple graphs may be superimposed in the same figure, each one specified in a different row. This is not the only way to plot multiple graph in the same figure. In fact, for each row there are different graphs which may be activated (each one labelled in a different category). They are: contour, scatter, streamlines, vectors and geometry. All these graphs can be set in one row or multiple rows. The reason for choosing the multi-row approach is if more graphs of the same type must be plotted in the same figure. For example, if the vectors of two different flow field velocities must be compared one above the other, they must be indicated in different rows, since only an arrow colour can be specified per row, and otherwise the two vectors type will not be discernible.

Another important feature is the variable fPlotOrder, which allows to specify in which order the graphs have to be superimposed when more than one graph is activated in the same row. For this purpose: 1.contour, 2.scatter, 3.streamlines, 4.vectors, 5.geometry. If for example, in one row of the input file the contour, vectors and geometry are activated (cContourAct = 1, vVectorAct = 1 and gGeometryAct = 1) and at the same time fPlotOrder = 514, the geometry will be plotted as first, then the contour and finally the vectors. If differently fPlotOrder = 145 the geometry will be plot as last.

List of all input variables with explanation - 2Dplotter

The variables indicated with [] allow single numbers, the ones indicated with {} are arrays (containing string of texts or multiple numbers)

```
%% Figure settings
fFigureNr = [];
fAxisLabelX = {};
fAxisLabelY = {};
fAxisLabelZloc = {}; %Set z label in locationPlotter and 2D3Dplotter
fTitle = {};
```

```
fAxisLimX = {}; %[Min;Max] works only if both Min and Max are specified (if
"-inf" or "inf" is set the limit will be the smallest or largest point)
fAxisLimY = {}; %[Min;Max] works only if both Min and Max are specified (if
"-inf" or "inf" is set the limit will be the smallest or largest point)
fAxisLimZloc = {}; %[Min;Max] set z range in locationPlotter and
2D3Dplotter, works only if both Min and Max are specified (if "-inf" or
"inf" is set the limit will be the smallest or largest point)
fLogAxis = {}; %[x-axis;y-axis;z-axis] logarithmic axes. 0: linear, 1: log
fInterpreterText = {}; % ['latex','none']
fShowFigure = []; %Show figure in interactive window [0,1]
fKeepAspectRatio = []; %Keep costant scaling in figure axes [0,1]
fGrid = {}; %[x-axis;y-axis;z-axis] Show axis grid in figure (0,1)
fGridMinor = {}; %[x-axis;y-axis;z-axis] Show axis minor grid in figure
(0,1)
fTickLabel = {}; %[x-axis;y-axis;z-axis] Show axis tick labels (0,1)
fFontSize = [];
fPlotOrder = []; %define the order of the layers plot in each figure.
                 %Each digit indicate a different plot and the number
                  define their order [e.g. 12345]
                 %1.contour, 2.scatter, 3.streamlines, 4.vectors,
                  5.geometry
fFlipNormal = {}; %(flipNormalFittingPlane;flipNormalRefPlane) Since any
plane can have two normals (same direction but opposite sign),
%sometime the figure can appear upside down or with left and right side
swapped. Activating the flipping
%allow to view it correctly, by manually changing the sign of either the
normal to the fitting plane or to the reference plane (xy).
fReverseAxis = {}; %(reverseXaxis, reverseYaxis) Reverse the figure axes,
to be used in conjuction with flipnormal to obtain the desired axes
orientation. N.B. it reverses the axis by changing its orientation (hence
negative towards right, instead of left). At the same time the axis labels
are changed of sign accordingly (while the data remain unaltered)
fVectorRotations = {}; % (angleX;angleY;angleZ) angles (in degrees) of
rotation to align the vectorial quantities (e.g. velocity) with the
measuring reference system. The rotation will be done in the order around
axis Z, Y and as last X
                                     % To be used in case e.g. the LDA is not
align with the model coordinate system and streamlines or velocity vectors
want to be plotted
%% Contour and scatter plot settings (1 and 2 in fplotOrder)
cContourAct = []; % Activate contourPlotter [0,1] N.B. It is used also to
activate the contourplot in 2D3D plotter
cScatterAct = []; % Activate scatterPlotter [0,1]
cWriteContourData = []; % Activate contour data file writing [0,1]
cdColormap = {}; %Colormap style. It can be a Matlab predefined style (e.g.
'parula','jet',etc.), or a user made one (e.g. [1 1 1;0.6 0.6 0.6; 0.4 0.4
0.4; 0 0 0] for a grey scale, colors are given in RGB format)
cdMethodInterp = {}; %['nearest','linear','cubic','natural','v4'] (it
affects also to streamlines)
cdSpaceInterp = []; %Interpolation grid distance (it affects also to
streamlines)
cLevNr = []; %Number of levels in colorbar
cIncludeMin = []; %[0,1] %0 if cdColorbarRange is specified, colormap
levels are spaced according to cdColorbarRange (and kept the same for each
graph)
                                         %1 same as 0 but including minimum value
before cdColorbarRange(1) in order not to have white patches if min is not
inside range
cdColorbarLabel = {};
```

```
cdColorbarRange = {}; %[Min;Max] works only if both Min and Max are
specified, otherwise the range is decided automatically from the min to the
max of each graph
cdColorbarLogarithmic = []; %[0,1] note: logarithmic scale cannot accept
negative values
cScatterPointSize = [];
cLineStyle = {}; %['-','--',':','-.','none'] Contour line style

%% Strealines plot settings (3 in fplotOrder)
tStreamlineAct = []; % Activate streamlinesPlotter [0,1]
tStreamlineDensity = []; %Density of streamlines [e.g. 1]
%cdSpaceInterp = []; %Modified from Contour settings
%cdMethodInterp = {}; %Modified from Contour settings
tStreamlineColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
tStreamlineWidth = [];
tShowArrows = {}; %['arrows', 'noarrows'] Show arrows on streamline

%% Vectors plot settings (4 in fplotOrder)
vVectorAct = []; % Activate vectorsPlotter [0,1]
vVectorAutoscale = {}; %['off','on'] Autoscale vectors automatically to fit
into the figure
vVelMag = []; %Magnification factor for vectors, if 1 no magnification
(used only if vVectorAutoscale is off)
vLineWidth = [];
vColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal

%% Geometry plot settings (5 in fplotOrder) N.B. Some variables are shared
with the location and 2D3D ploter (marked as gld)
gGeometryAct = []; %Activate geometryPlotter [0,1] N.B. It is used also to
activate the geometry in 2D3D plotter and locationPlotter
gldGeometryFilename = {}; %e.g. Geometry1.stl. N.B. Geometry must be in STL
format (locationPlotter and 2D3D plotter accept both ASCII and binary STL
format while for the section slicing only the binary is accepted)
gldGeometryTranslations = {};% (X;Y;Z) Translation which must be apply to
the stl geometry to align its origin with the plot data origin
gldGeometryRotations = {};% (angleX;angleY;angleZ) angles (in degrees) of
rotation to align the stl geometry  with the measuring reference system.
The rotation will be done in the order around axis Z, Y and as last X
gGeometryPlanarPoints = {}; %Three non colinear points laying on the
desired plotting plane (e.g. [0 -100 100;0 -100 0;100 -100 0]). This
variable can be used to force the selection of the slicing plane to be
different from the real one
gGeometryOversizeMargin = []; %(e.g. 0.1 for 10% of axes range) When the
range of the plot is chosen (by using fAxisLimX,Y,Z) the stl points of the
geometry outside this range are automatically excluded to save time in
making the slice.
                                        %However, if a geometric element
extends from inside to outside this range, the latter may not be correctly
shown. To overcome this issue a value here can be specified as a percentage
of the range in which the stl points will still be included.
gldFaceColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
gldEdgeColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
gldFaceTransparency = []; %Face surface transparency value from 0 to 1
(where 0 is invisible)
gldEdgeTransparency = []; %Edge transparency value from 0 to 1 (where 0 is
invisible)

%% Location plot settings N.B. Some variables are shared with the 2D3D
plotter (marked as ld)
lLocationAct = []; %Activate locationPlotter [0,1]
```

```
ldListDatafileShown = {}; %(DataNr1;DataNr2;...)List of datafile nr. whose
points are shown in the location plotter
ldViewAngles = {}; %(az;el) Set the viewing angles for the figure in
degrees (azimuth and elevation angles)
lMarkerType = {};
lMarkerColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
lMarkerSize = [];
ldRefPointCoord = {}; %(x;y;z) Coordinates of reference point (e.g. source
location) (leave blank if not needed)
ldRefPointColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
(leave blank if not needed)
ldRefPointSize = []; % e.g. 20 (leave blank if not needed)
ldRefArrowBaseCoord = {}; %(x;y;z) Coordinates of reference arrow base
point (e.g. wind direction) (leave blank if not needed)
ldRefArrowComponents = {}; %(x;y;z) Lengths of reference arrow in each
direction (leave blank if not needed)
ldRefArrowColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
(leave blank if not needed)
ldRefArrowSize = []; % (e.g. 2) Specify the arrow line width and head size
(leave blank if not needed)

%% 2D3Dplotter
d3DAct = []; %Activate 2D3D plotter [0,1] N.B. Used in conjunction with
gGeometryAct and cContourAct

%% Singleplotter settings
sSavePlot = []; % Save single plot [0,1]
sFormatFig = {}; % ['fig','png','jpeg','bmp','pdf','meta',etc.]
sFigRenderer = {}; % ['opengl','painters'] painters produces vectorial
figures, opengl rasterised figures
sLeftEdge = []; %in cm (This value is ignored when saving a figure to a
nonpage format, such as a PNG or EPS format.)
sBottomEdge = []; %in cm (This value is ignored when saving a figure to a
nonpage format, such as a PNG or EPS format.)
sPlotHeight = []; %in cm
sPlotWidth = []; %in cm

%% Multiplotter settings (Only values set in the first line of the input
file are used)
mPlotAct = []; % Save all plots in one file [0,1]
mFormatFig = {}; %['pdf','emf','png']
mResolution = {}; %works only if Renderer is opengl (e.g. r400)
mMainTitle = {}; %To be shown on top of the page
mGraphsxColumn = []; %Set the figure grid in the multiplot
mGraphsxRow = []; %Set the figure grid in the multiplot
mLeftEdge = []; %This variables control the spacing between figures (in cm)
mRightEdge = [];
mTopEdge = [];
mBottomEdge = [];
mSpaceX = [];
mSpaceY = [];
mPlotHeight = []; %Page height in cm
mPlotWidth = []; %Page width in cm
mMainTitleFontSize = []; %Fontsize of mMainTitle
mFontSize = []; %Fontsize of all the other text (replace fFontSize only in
the multiplot)
mInterpreterText = {}; % ['latex','none'] Applies only for the mMainTitle
mFigRenderer = {}; % ['opengl','painters'] painters produces vectorial
figures, opengl rasterised figures
mSingleColorbar = []; % [0,1] Show only the colorbar of the first graph
superimposed at the desired location (works only if colorbar range is set
for the first figure, whose properties are automatically extended to all
the other figures)
mSingleColorbarOrientation = {}; %['Vertical','Horizontal']
mSingleColorbarLocation = {}; %[leftEdge,bottomEdge,width,height] location
of the centre of the colorbar (where 0 is the right paper edge, 1 the left
edge) if mSingleColorbar = 1
```

Input files – LPplotter

The input file for the location plotter works as the 2Dplotter one. Different rows can be used to specify different line point properties or geometry colours. However, only one figure can be produced with one input file. For this reason the multiplot feature is not available and only the singleplot works.

List of all input variables with explanation - LPplotter

The variables indicated with [] allow single numbers, the ones indicated with {} are arrays (containing string of texts or multiple numbers)

```
%% Figure settings (To be specified only in the first line of the input
file)
lineNr = [];
fAxisLabelX = {};
fAxisLabelY = {};
fAxisLabelZ = {};
fTitle = {};
fAxisLimX = {}; %[Min;Max] works only if both Min and Max are specified (if
"-inf" or "inf" is set the limit will be the smallest or largest point)
fAxisLimY = {}; %[Min;Max] works only if both Min and Max are specified (if
"-inf" or "inf" is set the limit will be the smallest or largest point)
fAxisLimZ = {}; %[Min;Max] set z range in locationPlotter, works only if
both Min and Max are specified
fInterpreterText = {}; % ['latex','none']
fShowFigure = []; %Show figure in interactive window [0,1]
fKeepAspectRatio = []; %Keep costant scaling in figure axes [0,1]
fGrid = {}; %[x-axis;y-axis;z-axis] Show axis grid in figure (0,1)
fGridMinor = {}; %[x-axis;y-axis;z-axis] Show axis minor grid in figure
(0,1)
fTickLabel = {}; %[x-axis;y-axis;z-axis] Show axis tick labels (0,1)
fFontSize = [];
fViewAngles = {}; %(az;el) Set the viewing angles for the figure in degrees
(azimuth and elevation angles)

%% Location plot settings (These settings are the only which can be
specified for each input file line)
%These variables define the geometry properties. If the user wants to load
different geometries, these properties have to be indicated for as much
lines as the number of the geometries
lGeometryFilename = {}; %e.g. 'Geometry1.stl'. N.B. Geometry must be in STL
format (binary or ASCII). If this variable is indicated in the line, also
the other referring to the geometry must be indicated
lGeometryTranslations = {};% (X;Y;Z) Translation to apply to the stl
geometry to align its origin with the plot data origin
lGeometryRotations = {};% (angleX;angleY;angleZ) angles (in degrees) to
rotate to the stl geometry to align its axes with the plot data axes. The
rotation will be done in the order around axis Z, Y and as last X
lFaceColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
lEdgeColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
```

```
lFaceTransparency = []; %Face surface transparency value from 0 to 1 (where
0 is invisible and 1 solid)
lEdgeTransparency = []; %Edge transparency value from 0 to 1 (where 0 is
invisible and 1 solid)
%These variables refer to the measuring point markers
lMarkerType = {};
lMarkerColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
lMarkerSize = [];

%This variables set the reference point and arrow. Indicate only if a
%reference point and arrow are needed. Multiple points and arrow can be
%indicated (one for each line)
lRefPointCoord = {}; %(x;y;z) Coordinates of reference point (e.g. source
location) (leave blank if not needed)
lRefPointColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
(leave blank if not needed)
lRefPointSize = []; % e.g. 20 (leave blank if not needed)
lRefArrowBaseCoord = {}; %(x;y;z) Coordinates of reference arrow base point
(e.g. wind direction) (leave blank if not needed)
lRefArrowComponents = {}; %(x;y;z) Lengths of reference arrow in each
direction (leave blank if not needed)
lRefArrowColor = {}; %['r','g','b','c','m','y','k','w'] or hexadecimal
(leave blank if not needed)
lRefArrowSize = []; % (e.g. 2) Specify the arrow line width and head size
(they have to be the same) (leave blank if not needed)

%% Singleplotter settings (To be specified only in the first line of the
input file)
sSavePlot = []; % Save single plot [0,1]
sFormatFig = {}; % ['fig','png','jpeg','bmp','pdf','meta',etc.]
sFigRenderer = {}; % ['opengl','painters']
sLeftEdge = []; %in cm (This value is ignored when saving a figure to a
nonpage format, such as a PNG or EPS format.)
sBottomEdge = []; %in cm (This value is ignored when saving a figure to a
nonpage format, such as a PNG or EPS format.)
sPlotHeight = []; %in cm
sPlotWidth = []; %in cm
```

Input files – 7HP

The 7HP does not have a dedicated input file since it requires only two variables. These are inserted in EnFloSoftwareMatlabSuiteSettings.xls and are ngrid and OutputfileDecimal_Places. The first controlled the resolution of the calibration matrix, the second the number of decimal places in the output file.

Data files

Data files are tab-separated files used to load the graphs points in the software. They have to be placed in the data folder inside the folder of each part of the software. The name of the datafiles is Data#.xls where # is the number of the row in the input file to which they refer to. In the data files the data is organised in columns. The first four rows are for labelling only and are not read by the software. Note, differently from the input file, what written in the first row is not used as variable name in the code.

Data files – 1Dplotter

In the first column the profile number must be specified. This number indicate the profile in the figure which the point refers to. The following three columns report the location in which the point was acquired. The fifth and sixth column report the data considered in the x and y axis of the graph, respectively. If for example, the profile is a vertical profile in which the y-axis is the z coordinate, the sixth column may be equal to the forth (if the normalization is kept the same). Note that the information on the point location in the second to forth columns is not used in the profile generation, but only in the location plotter graph (if requested) to show the measuring points location with the geometry. The columns from 7 to 10 are dedicated to the errorbar data. If an errorbar cell inside the column is left empty, the relative error for that point will not be shown. The errorbar can be specified for the x-axis, the y-axis or for both the axes. Moreover, the errorbar can be symmetrical, with the upper limit equal to the lower (in this case only one number has to be indicated per axis) or with different upper and lower limits. The format is controlled in the input file with the variables fErrorbarColumn and fErrorbarOrientation. The first indicate if lower and upper error are indicated by either 1 number or different 2 values. The second indicate if the errorbar is horizontal (hence referring to the x-axis), vertical (y-axis) or both. The following combinations are possible:

fErrorbarColumn = 1

- if fErrorbarOrientation = horizontal or vertical the errorbar is indicated in the 7th column only and represents the lower and upper limit.
- If fErrorbarOrientation = both the errorbar is indicated in the 7th and 8th column and represent the lower and upper limit of the x-axis (7th) and y-axis (8th).

fErrorbarColumn = 2

- if fErrorbarOrientation = horizontal or vertical the errorbar is indicated in the 7th and 8th columns and represent the lower (7th) and upper (8th) limit.
- If fErrorbarOrientation = both the errorbars are indicated in the 7th, 8th, 9th and 10th column and represent the lower (7) and upper (8) limit of the x-axis, and the lower (9) and upper (10) limit of the y-axis.

Data files – 2Dplotter

In the 2Dplotter the first 3 columns of the data file are for the x, y and z acquisition locations of the points. The 4th and 5th columns are for the quantities to show in the x and y axes of the 2D graph. If the latter two columns are left empty, the point locations specified in the first 3 columns will be used to compute a fitting plane and the quantities of the two axes will be automatically computed. The possibility to manually set the axis quantities for each point, independently from the location of acquisition, is particularly useful in case they are not a geometric quantity (as in the case of a graph in which e.g. the x-axis is a temperature, the y-axis a velocity and the contour a concentration).

It should be stressed, that if the 4th and 5th columns are left empty and a calculation is computed to find the fitting plane of the given points, a plane will always be found (provided that the points are more than 2 and not collinear) even though they might be not coplanar. In case at least one of the points is more than 1 (assuming millimeter as measuring unit) far from the computed fitting plane, a warning message will be issued from the software in the output string and saved in the error log file. Nevertheless, the computation will continue forcing all the points to stay in the fitting plane. A check of the fitting plane respect to the real measuring locations can be visually performed by plotting a contour of the points together with the measuring locations in a 3D graph, as done in the Example 2DEx02.

The 6th column is dedicated to the scalar quantity which is used in contour graphs or scatter graphs. This can be e.g. a concentration, a temperature, a component of a vector, etc. If no contour or scatter plots have to be performed, this column can be left empty.

The columns from 7 to 9 are for the vector quantities to be used in the plotting of streamlines or vectors. The 7, 8 and 9 columns quantities are considered aligned with the axis of the location points. In case this is not true, the input variable fVectorRotations can be used to indicate the angles of rotation between the reference system used for the locations and the one for the vector quantities acquisition.

N.B. Only 2D streamlines and vectors can be plotted (the 3D version has not been implemented yet). This means that if all the 3 components of the vector quantity are given, they should be rotated to be aligned with the fitting plane (such rotation is different from the one given by fVectorRotations, in fact, that one is used to align the vectors with the reference system of the location, while this one to align the vectors with the fitting plane reference system). In order to compute such rotation a fitting plane must be calculated as well (hence the 4th and 5th columns must be left empty).

In case, the user wants to directly assign the vector quantities without rotate them to the fitting plane reference system, he should leave the 9th column empty (without even specifying the header). In this way the 7th column will be automatically considered aligned with the horizontal axis and the 8th with the vertical axis of the final graph and used as they are.

Data files – LPplotter

The LPplotter data file is very easy, since only three columns with the location coordinates must be given. If the user wants to show the points with different colours or properties, more data files should be created, each one with points sharing the same properties.

Data files – 7HP

The 7HP requires two data files, one is the calibration file, whose name has to be unique since identify the calibration. The second is the PressureResultsFile.xls, which contains the pressures taken with the seven-hole probe.

For both the files the association is done on the basis of the column location (as for all the data files), and not the header name (as for the input files). So it is important to not change the location of the columns. Moreover, the data starts to be read from the fifth row in the file.

The important data for the calibrations is the first 9 columns, which contains, in the order: Yaw (deg), Pitch (deg), P0 (Pa), P1 (Pa), P2 (Pa), P3 (Pa), P4 (Pa), P5 (Pa), P6 (Pa), Pstat (Pa), P_ref (Pa).

The important data for the pressure files is the first 9 columns, which contains, in the order:

Time (s), 7HP.viCOM71P0, 7HP.viCOM71P1, 7HP.viCOM71P2, 7HP.viCOM71P3, 7HP.viCOM71P4, 7HP.viCOM71P5, 7HP.viCOM71P6, density (kg/m^3)

Other columns can be added after, but the data will not be used.

Examples explanation

A series of examples have been created to illustrate the various functionalities of the software. An executable file EnFloSoftwareMatlabSuiteEmulator.exe has been created which allows to automatically copy and paste the files from a folder (EnFloSoftwareMatlabSuiteEmulator\) and run all the examples one after the other. This is used also to test the software every time a modification is done. A log file (EmulatorLog.txt) with the time it took to run each example is created, as well.

In the following the examples will be briefly introduced.

1Dplotter

The 1Dplotter is thought to produce plots of profiles with the maximum flexibility, so that each profile in each figure is controlled independently by the others.

Example 1 – Eight vertical profile graphs + Errorbars + location plot

In this first example a large range of functionalities of the 1Dplotter is explored (as shown in Figure 1). 5 vertical profiles at 5 different locations are shown in each of the 8 graphs. Note that the software is organised so that each figure in a multiplot can have a different number of profiles and with different legend properties. A single legend is shown in the bottom in a row style. Its location and number of columns can be manually modified in the input file. Also a legend title can be set, “x (mm)” in this case, by using the mLegendTitle variable. The legend is automatically created by checking all the profiles properties and showing all the different ones only once (in the order they appear in the multiplot).

The distance between the graphs and between the graphs and the edges can be manually specified, as well (see the “m” variables in the input file). For what concerns the axis labels, for each figure the user can specify an x-axis label, y-axis label and title. If the user does not want to show some of them, their space in the input file must be left empty. In this example, only the title is shown for each figure, while the y-axis label is shown only for the figures close to the left edge and the x-axis label is never displayed.

The label and in general the text can be inserted using two different interpreters (which must be specified for each figure). If “none” is used as text interpreter no special character can be employed in the text. Differently, if “latex” is used, all the latex characters can be used, including the italic and bold. In this case every text string must start and finish with $ (as shown in the example).

All the possible types of errorbar are shown in the first six graphs for one point only each time. The types are: Figure 1 – errorbar horizontal, not-symmetrical, Figure 2 – errorbar horizontal symmetrical, Figure 3 - errorbar vertical not symmetrical, Figure 4 - errorbar vertical symmetrical , Figure 5 - errorbar both not symmetrical , Figure 6 - errorbar both symmetrical.

Finally, also the location plot is shown. The colour and marker styles are automatically assigned from the profile properties it refers to. An arrow and a point can be also added to the figure, in this example they represent the free-stream wind direction and the source. The orientation of the location plot can be manually defined by setting two angles in lViewAngles. They can be easily identified if the location plot figure is shown in the interactive window (fShowFigure = 1). By rotating the figure with the dedicated button in the window the two angles are shown in the bottom left corner. Finally, the location plot refers only to the indicated figures (specified in the variables lListFiguresShown). In this way, if the user wants more location plots can be shown in the same multiplot, each referring to different figures.

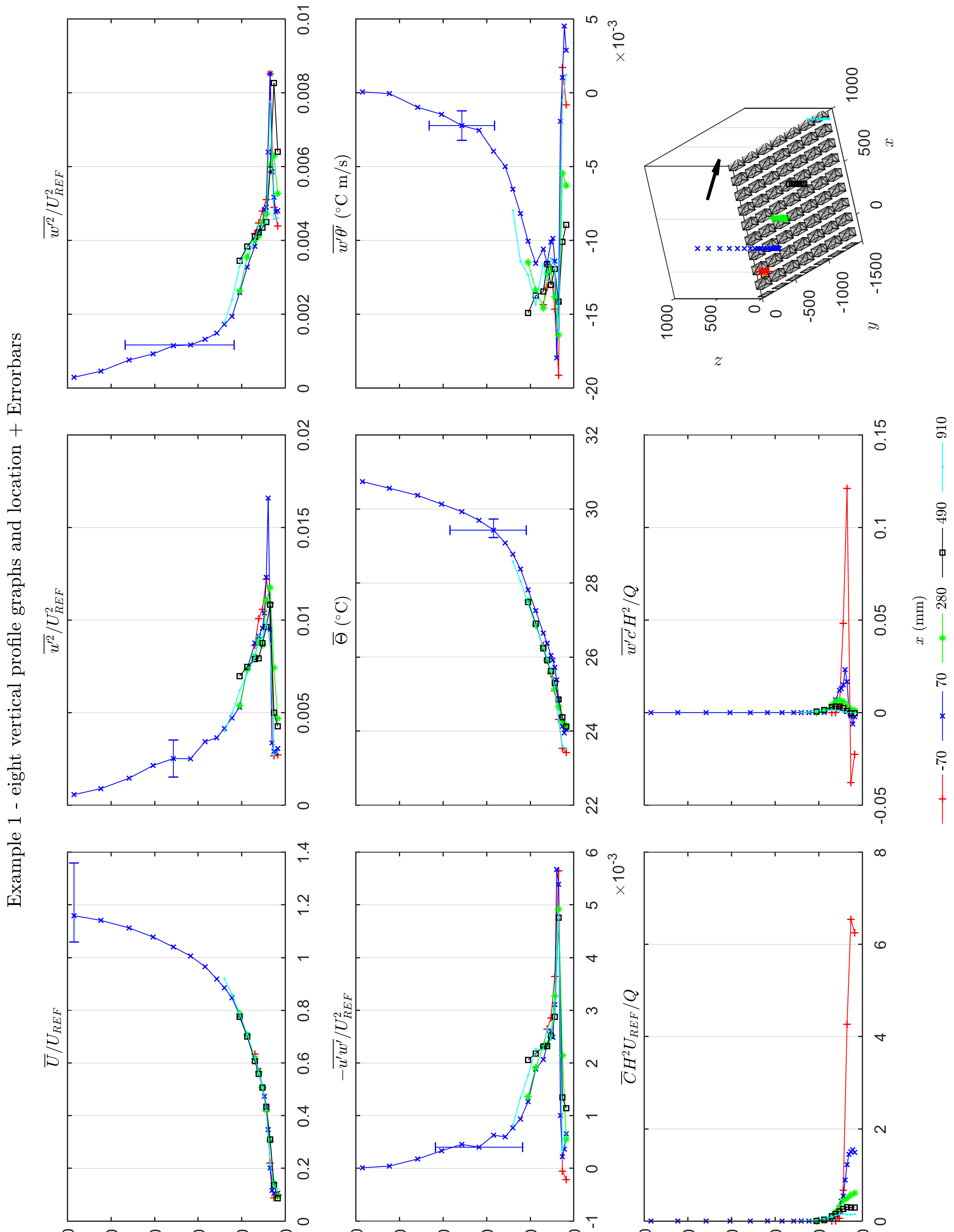

Figure 1: 1Dplotter example 1. Output image

Example 2 – Fitting features example

In this example, whose output image is shown in Figure 2, the fitting features are explored. In the upper left figure three profiles are shown, each of them fitted with the logarithmic wind law. The fitting for each profile is activated by the input variable pFitOn. The variable pFitIndepAxis indicates which is the independent axis (the possible options are X or Y). The fitting equation must be manually specified in pFitEquation. In the case of the example the log law is written as 0.064/0.4*log((Y - C(1))/C(2)). Y is the independent variable (being in this case the y-axis). C(1) and C(2) are the coefficient which will be found with the not linear least mean square fitting. The user can specify as many fitting coefficients as he wants, in the form of C(1), C(2), C(3), …, C(n).

The variable pFitStart allows to specify the starting points for the fitting. Each coefficient must have at least one starting point. It is also possible to indicate more starting points for the coefficients, so that the fitting will be repeated starting with each of them and the solution which guarantees the smallest error will be selected. The starting points have to be indicated as in the example: [2 3 4 5 6;10 10 10 15 20]. The numbers are delimited by [], the numbers 2 3 4 5 6 represent the starting points for the first coefficient, while 10 10 10 15 20 are the ones for the second coefficient. They are separated by a ;. The variables pFitLowerBound and pFitUpperBound specify the lower and upper limit of the independent variables, filtering the input data which will be used for the fitting. The limit can also be set as -inf;inf. pFitIndepAxisFitRange and pFitIndepAxisPlotRange, instead, indicate the range of the fitting profile which is shown in the figure. The fitting profile is shown as a line formed by 100 points, whose properties are taken from the one specified for the fitted profile. The latter is shown only by means of the markers, while the line is used for the fitting profiles.

In the right upper figure the same points are inserted in a single profile (all blacks) so that all of them together can be used to find a single fitting profile. The lower left figure, instead, represent a combination of the previous two. In fact, the profiles are plotted with different colours but not used for the fitting. In the same figure all the points are inserted again as a forth profile, whose markers are set as invisible (‘none’). Only this forth profile is used for the fitting curve, which is then equivalent to the one obtained in the upper right figure. The same is done for the last figure (bottom right) with a linear fitting in this case, equation C(1)*Y + C(2).

The coefficients obtained for each profile are shown in a file called FittingCoefficients.xls, together with the fitting error. Such file is created in the output folder if the fitting feature is activated for at least one profile.

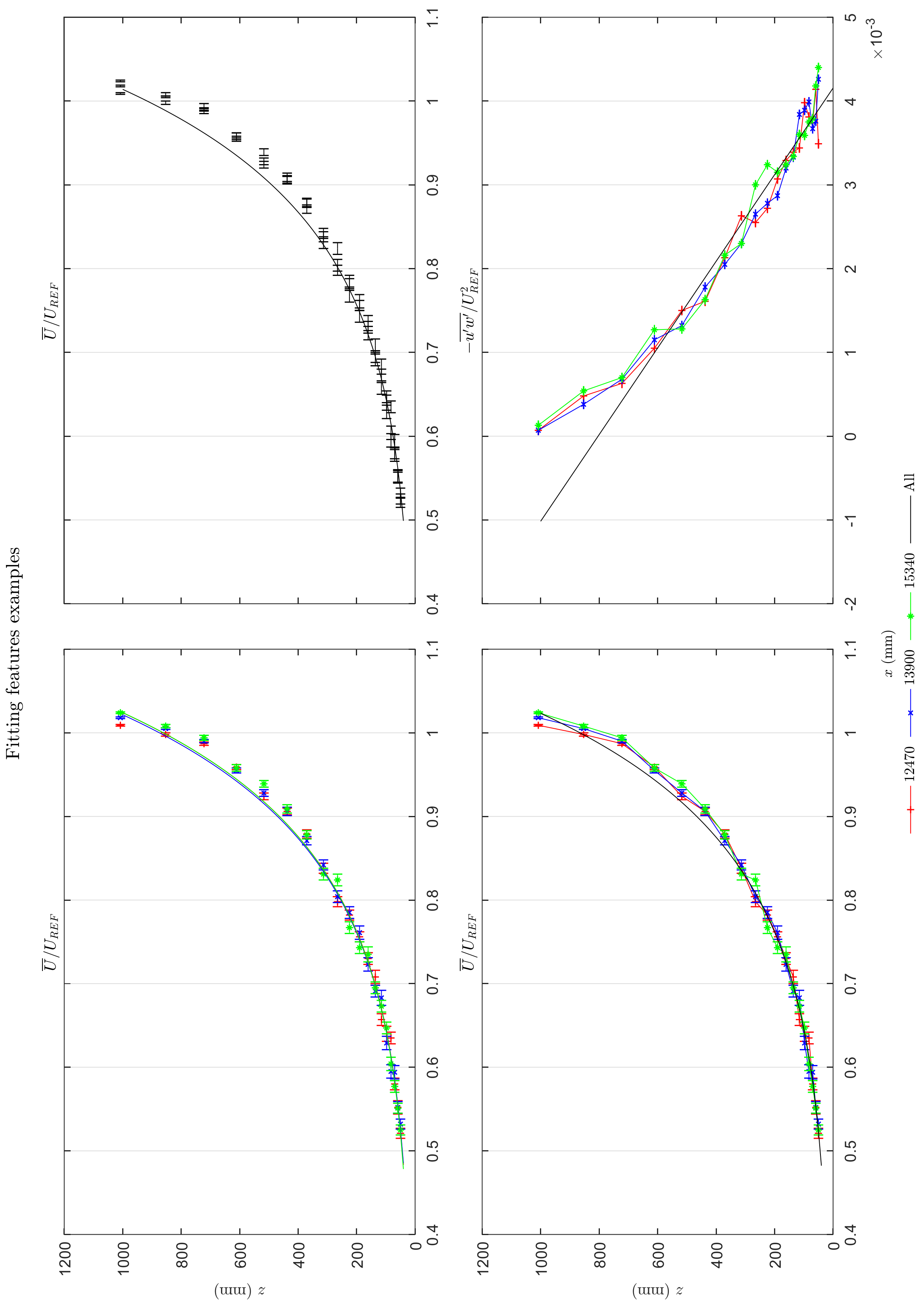

Figure 2: 1Dplotter example 2. Output image

Example 3 – Double y-axis example

In this last example on the 1Dplotter the focus is on the double y-axis feature. Each profile in each figure can be linked to the right y-axis if pAxisYright = 1. If at least one of the profiles in a figure is linked to the right y-axis, the axis is created. The label and range for this axis is controlled by the variable fAxisLabelYrightZloc and fAxisLimYrightZloc, respectively. The ticks (literally the numbers along the axis) can be controlled by the third number of fTickLabel, as well as the logarithmic feature with a third number added to fLogAxis. A colour for the right y axis must be specified in fYrightAxisColor. In the example it is blue as the profile associated with it (to help highlighting the association). The double axis feature is compatible with the fitting as well.

It should be noted that for a coding necessity, if the double axis feature is activated the interactive window will appear for each figure, even though not activated, and will disappear at the end. Unfortunately, this slows the code execution, but it was not possible to be fixed.

In this example the legend is shown as a single column on the right side of the multiplot.

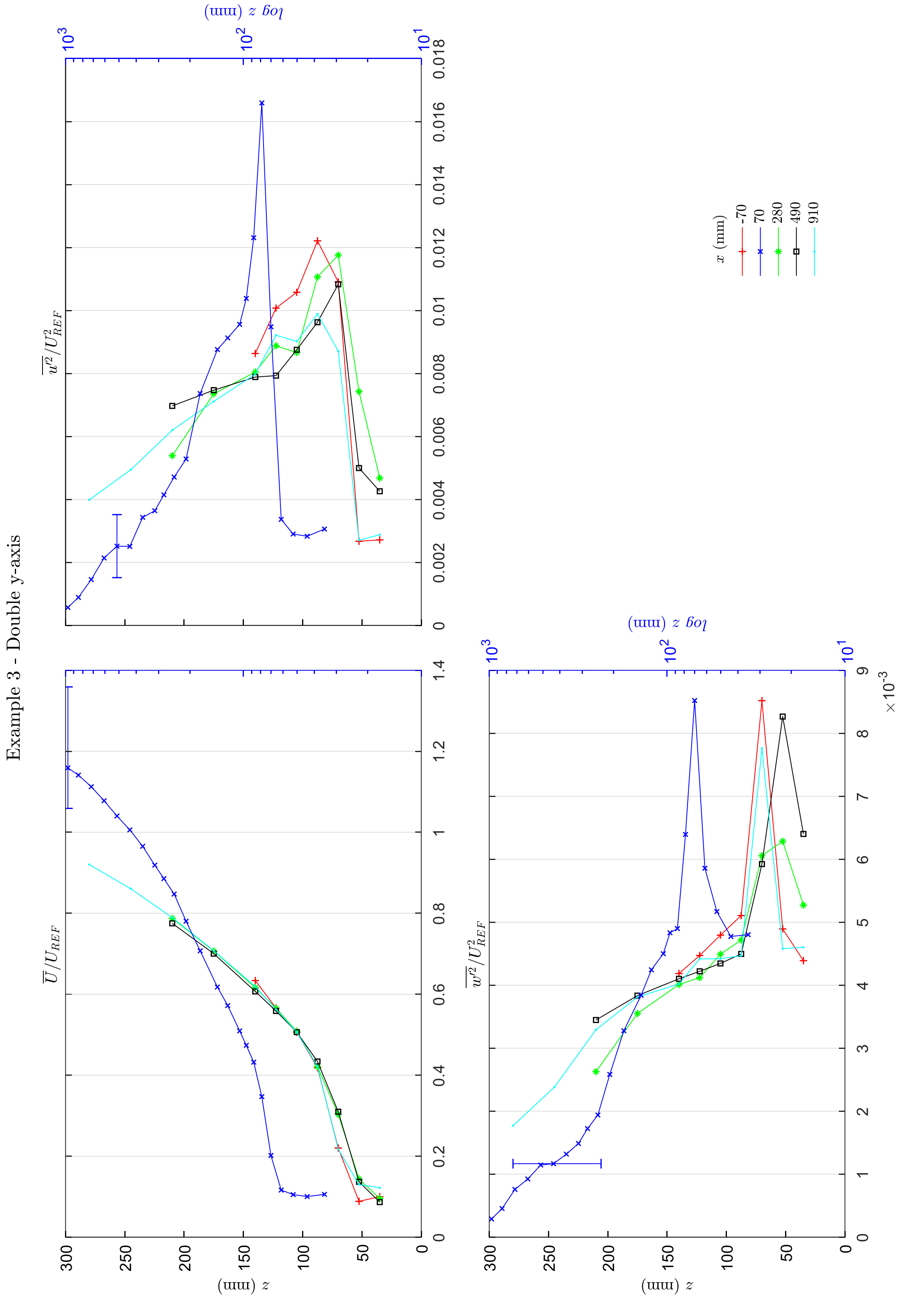

Figure 3: 1Dplotter example 3. Output image

2Dplotter

Example 1 – Contour locator 3D

In this example contour graphs of concentrations in two cross-sections are shown, before separately in 2D graphs and then together in a 3D graph. Also the measuring points are represented in the bottom left graph. The geometry in the 2D graphs is shown as a slice of the 3D stl file.

If the forth and fifth columns in the data files are left empty, as in this example, the software performs a fitting among the coordinates of the measuring points to find the equation of the fitting plane. A new axis reference system is defined on this plane and the measuring points are translated and rotated (if necessary) to be aligned with such plane, so that the z coordinate in the new reference system is zero for all the points. If the z coordinate is not zero, it means that the points are not coplanar. In this case a warning message is issued (if at least one z point coordinate is larger than 1), but the programme does not stop and the points are forced to lay on the same plane by imposing all z = 0. In the following the method is explain in more detail

As shown in Figure 5, there are two planes: the “fitting plane” and a “reference plane”, which is always the plane z = 0 (also called xy). The two reference systems are xyz for the reference plane and x’y’ for the fitting plane. Two possibilities can appear, depending on the fitting plane.

- The fitting plane is parallel to the reference plane. In this case no transformations are needed and x’ = x, y’ = y.
- The fitting plane is not parallel to the reference plane, and in the most general case is randomly oriented.

The difference between the two cases is not always easy to determine. A threshold has been introduced so that if the plane is not perfectly parallel, but still inside that threshold, it will be considered as parallel.

In the second case the method identifies a translation and two rotations to apply to the measuring points in order to align the fitting plane and the reference plane. More in detail, the intersection line equation between the two planes is computed, then the intersection between this line and either the x or y axis of the reference plane is chosen as origin of the x’y’ reference system (O’). The choice between the two is done considering the closest to the reference plane origin (smallest OO’ in Figure 5). Once the coordinates of O’ in the xyz reference system are found, all the measuring point coordinates are translated of the quantity $x - x_{O'}$ and $y - y_{O'}$. The angle existing between the fitting and reference plane is computed and a rotation of the measuring point coordinates around the intersection line of such angle is made. Now if the measuring points were all coplanar the z coordinate should be zero (or very close to zero) for all of them. At this point another final rotation may be necessary around the z’ axis (not shown in Figure 5) of the angle (possibly) existing between the x’ axis and the reference plane xy. After the last rotation is applied, the x’ axis will be coplanar with plane xy.

The described method allows, given a random set of more than 2 not collinear points in the 3D space, to always find a fitting plane and show them in a 2D reference system. Nevertheless, the resultant representation will be accurate only if the points were coplanar.

There is also another issue not yet debated. In fact, the result of the fitting of the points is the equation of the fitting plane. However, each plane can have two normal vectors, with the same direction but opposite sign. For example, the normal to the reference plane may be either [0 0 1] or [0 0 -1]. This uncertainty brings in some cases to undesired representations of the graphs. For this reason, if the user does not like the resultant figure orientation (which may appear, for example, upside down), he can flip the normal of either the fitting or the reference plane (or both together) with the input variable fFlipNormal. In some cases the fFlipNormal does not allow to obtain the desired figure orientation. For this reason, the variable fReverseAxis has been introduced, which allow to reverse either the x or y-axis (or both). In this way all the figures can be rotated respect to the previous orientation, while the axis label is not modified. This modification has effect also on the contour writing file.

What described until now is applied to the point coordinates every time the fourth and fifth columns of the data file are left empty. If they are not empty, then their data will be used as x' and y' information for the plotting and the point coordinate data will be ignored.

For what concerns the geometry in the 2D graphs, a slice of the 3D stl file has to be performed to show correctly the 3D geometry together with the measuring points in the same reference system.

To produce the slicing, a user-made software previously created for 3D printing has been modified for this purpose. However, such code can produce cuts only parallel to the xy plane, so if the desired slicing plane is not parallel to the xy, transformations have to be applied to the geometry points. This is done in the same way as described for the measuring points.

To identify the slicing plane for the geometry there are two ways: either the fitting plane can be used or the plane can be identified by three points which have to be manually given by the user in gGeometryPlanarPoints. The first approach is used in this example, but in case the second wants to be used, the three points must be not collinear and in the format e.g. [0 0 5; 10 1 0; 0 5 0], namely surrounded by [] and separated by ;. The second method is useful in case the user wants to produce a cut in the geometry in a different location from where the points where sampled. For example, if the sampling plane is above all the buildings, no geometry will be shown because no buildings are crossed by the fitting plane. But if the user wants to show the buildings as well in transparency, he can indicate manually a lower plane which crosses the buildings and so allowing to show them.

In the present example y' = z, while x' is a combination of x and y. Please note that there is an error in the sampling location of the contour plot in the upper right figure, caused by the fact that the points closer to the right-side building were not aligned with the others. In this way the resultant fitting plane is not coplanar with the points and not parallel with the previous contour plot. This has been used as example to show what happens if the points are not coplanar, but a fitting is attempted. This will be better shown in the next example.

The lower right figure shows the contours of the first two graphs together in the 3D geometry. This is possible thanks to the d3D package, which allows to convert back in 3D the contours already created in 2D. More in detail, the grid mesh for the contour is created in 2D, after all the transformation described previously, and then such transformations are applied back to the mesh to rotate and translate it back in 3D. In d3D package at the moment is only possible to show contour graphs and point coordinates (the latter shown in the lower left figure). Streamlines and vectors may be also possible but they have not be implemented yet.

Finally, a single colorbar is shown for all the figures. This feature is activated with mSingleColorbar = 1. mSingleColorbarOrientation and mSingleColorbarLocation then defines the colorbar orientation (vertical or horizontal) and the location in the page. To activate the feature the colorbar range for the first figure must not be empty. The colormap settings of the first figure are automatically assigned also to the others (namely cdColormap, cLevNr, cdColorbarRange and cdColorbarLogarithmic).

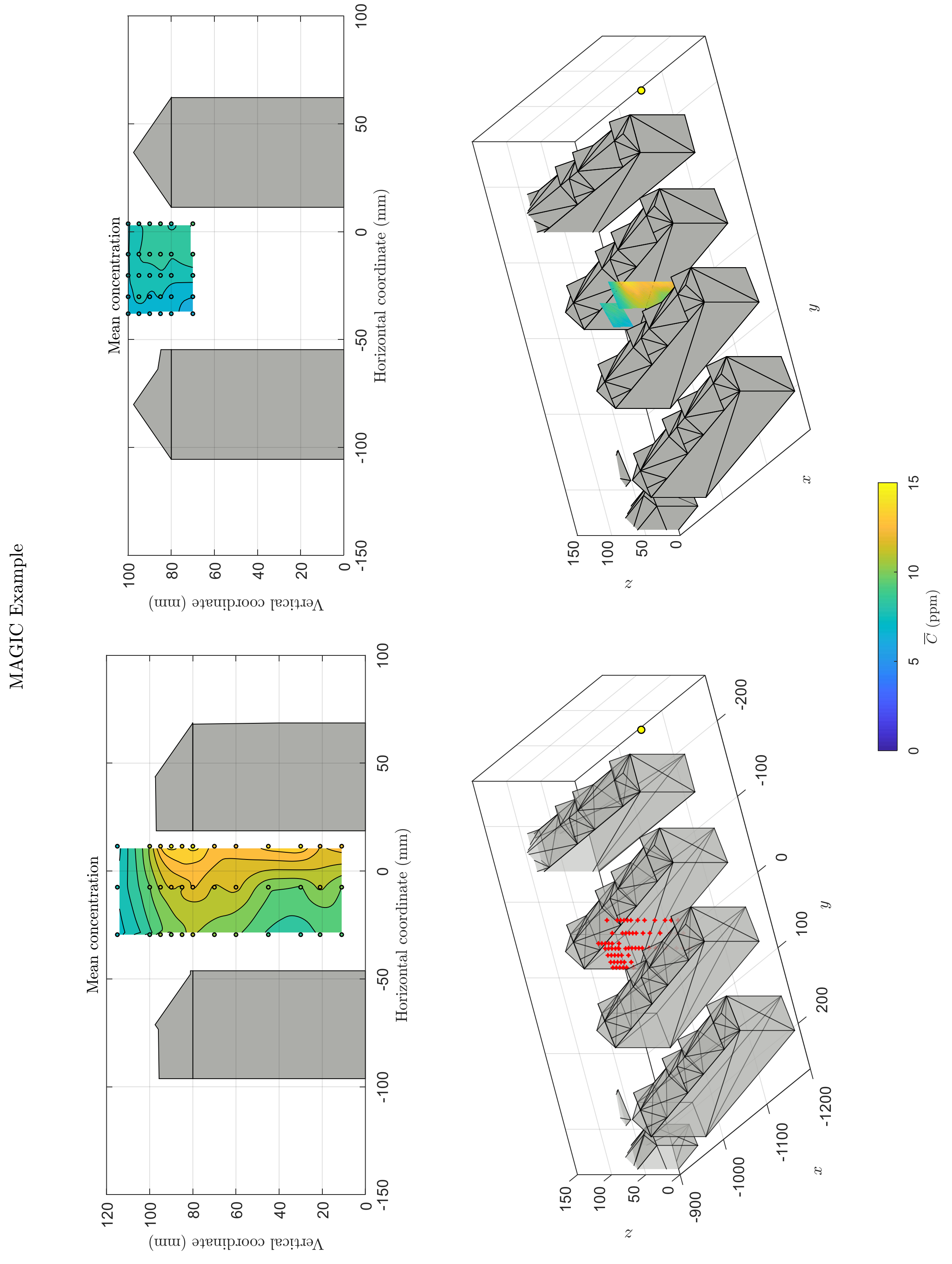

Figure 4: 2Dplotter example 1. Output image

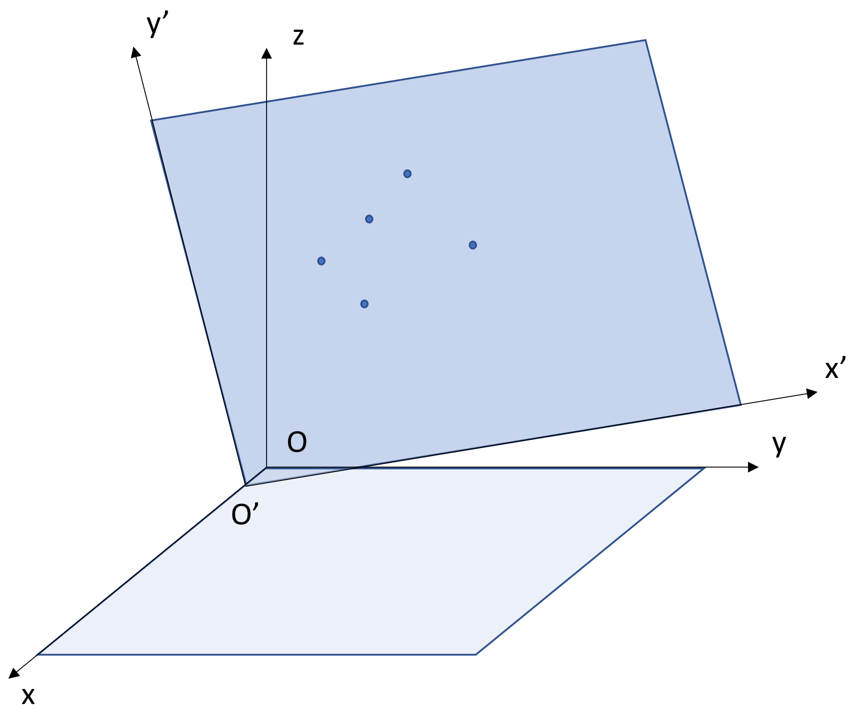

Figure 5: Slicing (dark blue) and reference (light blue) planes

Example 2 – Single 3D Contour locator

This second example reproduces the same case as the first one, but now only one picture is shown. This is to demonstrate how it is possible to show only one 3D picture made of multiple contours. The input file is composed by 3 instruction rows. In the first two the settings for the contours are given, but without graphs activated. The third is linked to the previous two by means of ldListDatafileShown, and controls the 3D graph, by activating d3DAct. The image is here created with the multiPlot by setting 1 row and 1 column. However, it can also be created with the singlePlot (which also in case of multiple figures allows to save them singularly).

In this example, the fact that the figure is larger allows to better appreciate the fact that in the second contour graph the contour plane is not align with the points, being the last vertical profiles out of the plane.

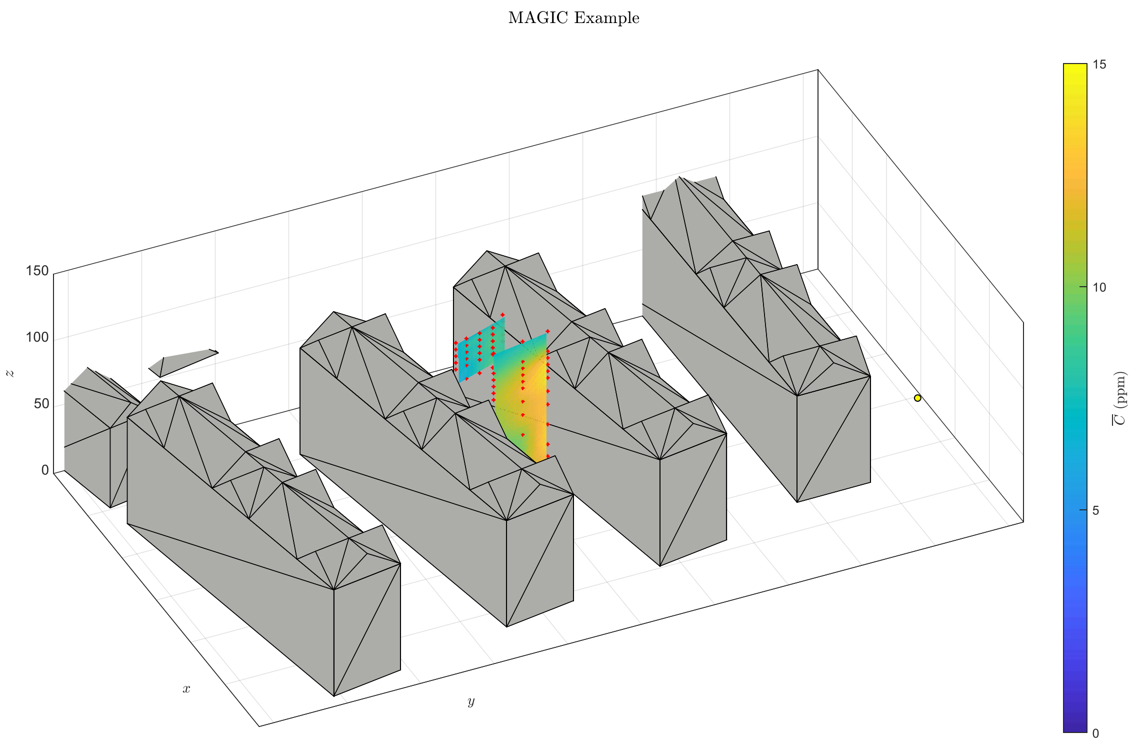

Figure 6: 2Dplotter example 2. Output image

Example 3 – Tunnel VS model coordinates

In this example, two contour plots of concentration are shown. They are taken from the same data but the first is in tunnel coordinates and the second in model coordinates (as also specified in the title of the two figures on the right side, which represent the d3D point locations plots, even though they appears like 2D picture, because they are seen from above.). They have been produced with the same geometry file. In the first case the measuring points were expressed in tunnel coordinates, hence in a reference system aligned with the wind tunnel, while the model was rotated of 45° respect to the vertical. In order to correctly represent the geometry in this case the stl has to be rotated of -45° (done through the variable gldGeometryRotations). The stl is also translated because the origin was wrongly placed (possible thanks to gldGeometryTranslations).

In the second plot, the model coordinates are used for the points and no rotation is applied to the geometry.

The geometry file is made of hundreds of buildings, which make it very heavy and quite slow to slice. For this reason, two features have been developed. Firstly, the first time the geometry is sliced, the result is stored in an array, so if the same slice is requested for another plot, it is taken from memory and not computed again. Secondly, if the range of the graph is manually defined (as in this example) the geometry points outside this range are removed and not considered in the slicing. However, in case of edge buildings for which a part is outside the range, the building would miss or not be correctly displayed. To solve this issue a threshold of the range can be manually defined by the user, inside which the points will still be considered. This is done by the variable gGeometryOversizeMargin. In this example a value of 0.1 is set, meaning that only the points farer than 10% the range from the edge will be filtered. This is sufficient in this case to show correctly all the buildings. If the buildings extended farer across the edges, a larger value might be considered.

The image can be saved in either vectorial format or rasterised. This is done by selecting "painters" for vectorial and "opengl" for the rasterised in mFigRenderer. Due to the large geometry, the vectorial format is very slow to be created in this case, so it is recommended to start the creation of the picture with opengl and set painters only at the end, when the picture is set up.

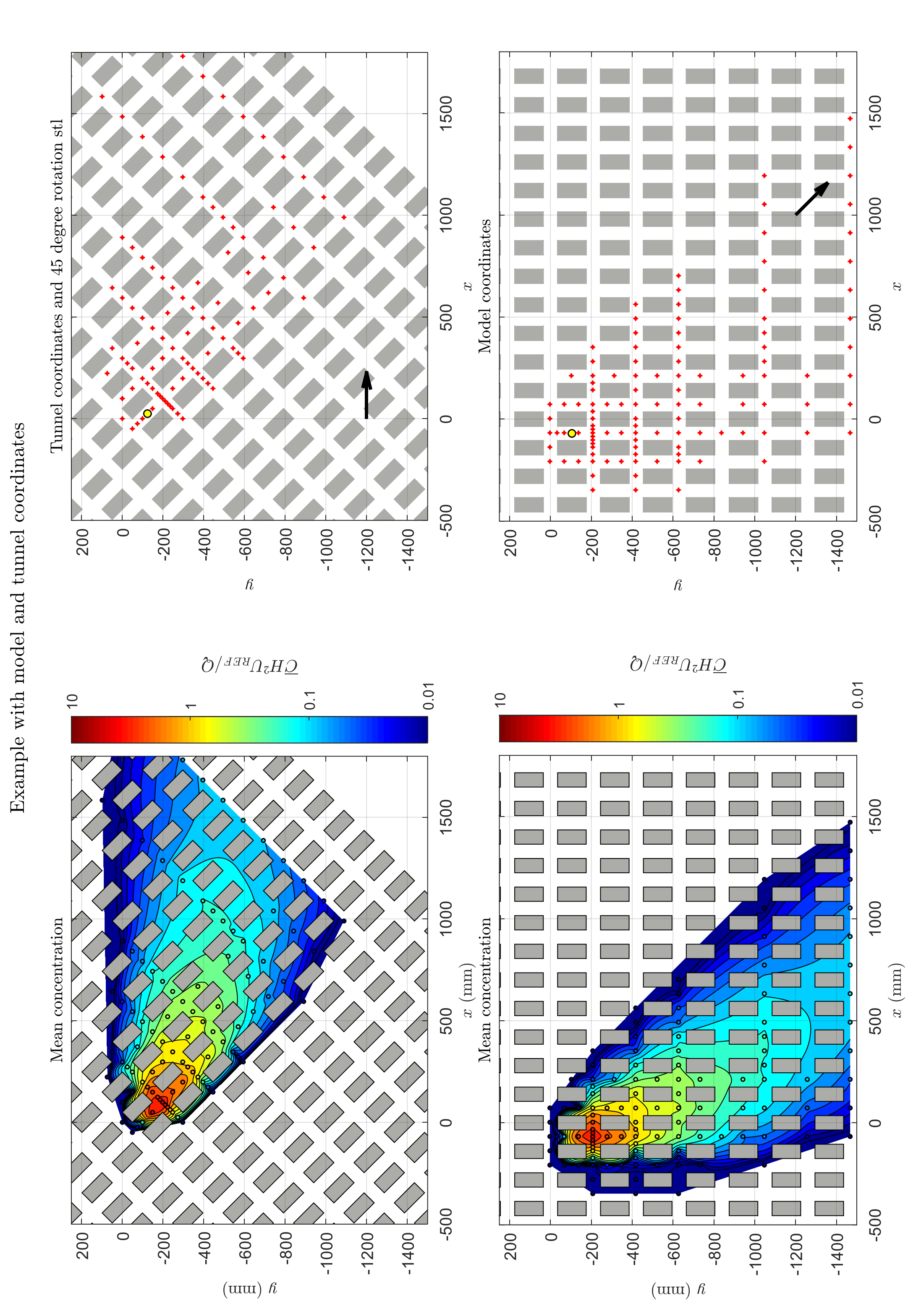

Figure 7: 2Dplotter example 3. Output image

Example 4 – Multiple plots in the same location - streamlines, and vector plots

In this last example vectors, streamlines and contours are shown in the same location in multiple graphs of different quantities. In the first figure the vectors are set up in the first input file row, while the scatter plot and the geometry in the second. This is to show the possibility of layering multiple graphs in the same figure starting from multiple rows in the input file, provided that they are linked to the same figure in fFigureNr. The same data file has to be repeated twice, though. One called Data1.xls and controlled by the first input file row, the second called Data2.xls and controlled by the second line.

In the third and fifth graph streamlines are plotted. For the third graph three components of the vectors are given in the column 7, 8 and 9 of the Data4.xls. In this way the rotation of the components is triggered, and they are rotated according to the fitting plane. The y component was not sampled, and it is set to zero. However, in this particular case the y component is perpendicular to the fitting plane and so it does not cause problems. In the fifth figure, instead. Only two components are given in the 7 and 8 columns of Data6.xls. In this way the rotation is not triggered, and these are shown as directly aligned with the horizontal and vertical axis, respectively.

In the fourth graph the contour has a logarithmic scale, set by cdColorbarLogarithmic. Also the horizontal and vertical axis can be set as logarithmic, if needed.

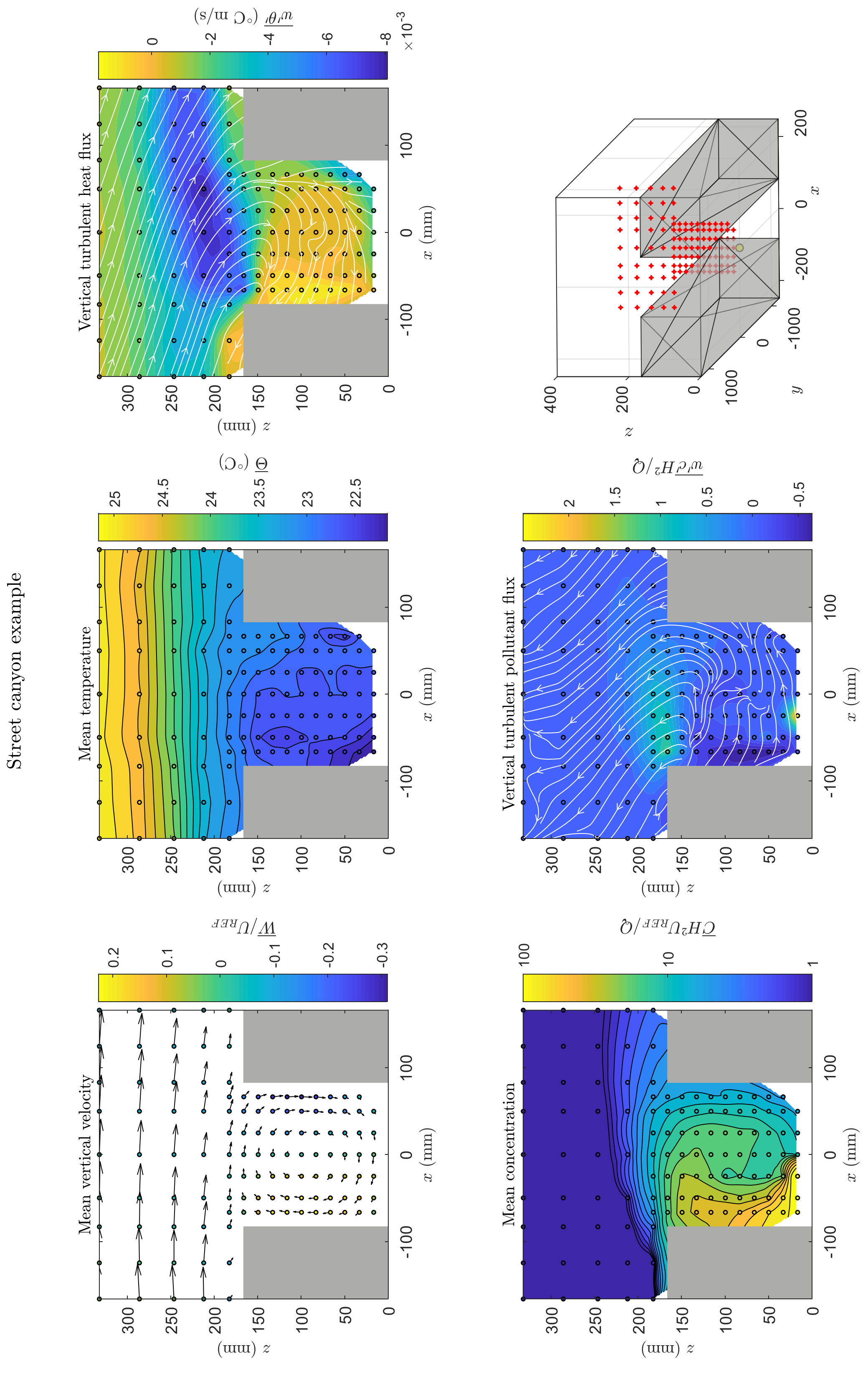

Figure 8: 2Dplotter example 4. Output image

LPplotter

Example 1 – Cross-section with three different colours for points and two for geometry

This example shows the feature of the location plotter stand alone. This is thought to display the measuring points while the acquisition is still going to show which points have already been acquired or are going to be sampled.

Multiple colours are possible for the points and the geometry, by using the multiple input file line feature also implemented in the 2Dplotter

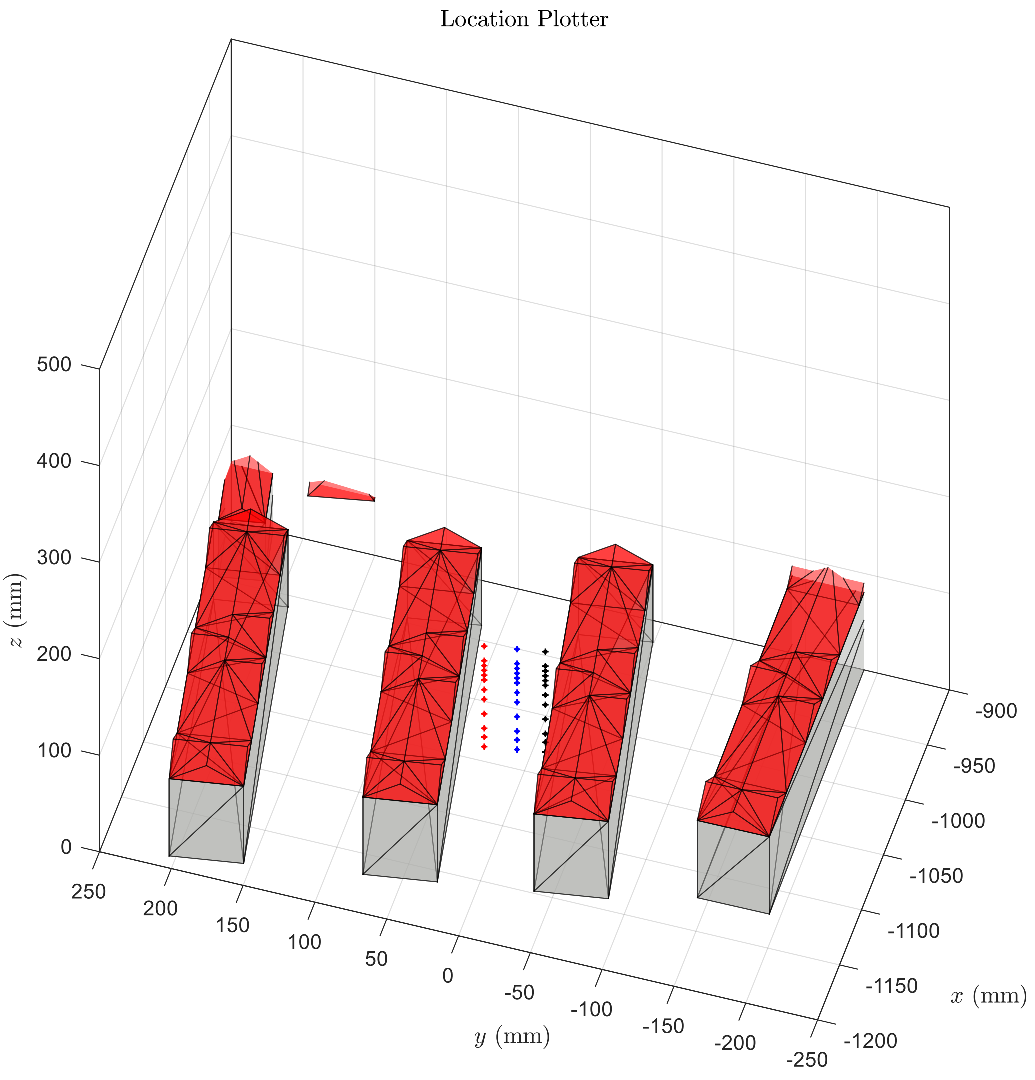

Figure 9: LPplotter example 1. Output image

7HPplotter

The example about the seven hole probe contains three points pressure to post-process. In particular, the third point has got the same calibration file as the first. This is to show the feature of the calibration memory. In fact, when a new calibration file (recognised by the name) is loaded, it is stored in an array and when the same calibration is asked again, instead of loading it another time, the one from the array is used, saving some seconds. All the employed calibrations then remain in the memory until the suite is closed.

List of modifications for each version

1Dplotter

%%%%%%%%%%%%% v1.0 main features %%%%%%%%%%%%%
12/04/2019
- produce 1D profile plots starting from a series of 3 to 5 column result file (column 4 and 5 are for the standard error)
- all inputs are given through an input file in which every row specifies the settings for each profile in each figure
- all input variables have preassigned default values in the code, so only the ones that you want to modify have to be specified in the input.xls file
- it can accept multiple result files and produce a separate plot for each and/or a single plot for all of the figure together
- an interactive window for each plot opens, which allows some operations
- legend works only for the single figure but not yet in the multiplot

%%%%%%%%%%%%% v1.1 main news respect to v1.0 %%%%%%%%%%%%%
16/04/2019
- deleted default value assignment in input file. Now the program only check if all the variable names are present in the input file
- The legend can be placed everywhere in the multiplot figure by specifying the location of the centre of the legend (range 0 - 1)
- The multiplot legend can be arranged in different columns
- errorbars are now implemented for both x and y axis

%%%%%%%%%%%%% v1.2 main news respect to v1.1 %%%%%%%%%%%%%
25/04/2019
- legend and labels plotted in single figure only if needed, to save time
- errorbar columns in result file to be inserted only if needed, to save time
- generated multiplot file automatically opened
- '/' replaced by '\' in file and folder name
- axis label ticks can now be deactivated for each graph independently

%%%%%%%%%%%%% v1.3 main news respect to v1.2 %%%%%%%%%%%%%
26/04/2019
- Once started programme remain active in a while loop for 5 minutes waiting for input.xls file updates. If it is updated a new plotting is started. If input.xls is deleted the programme stop
- two new input variables have been added for the errorbars (fErrorbarOrientation and fErrorbarColumn). The first describe the errorbar orientation (either 'none','horizontal','vertical' or 'both' are possible), the second specify the format (only one column with same upper and lower limit, or two columns with different lower and upper limit). Note that if both errorbar are plotted fErrorbarColumn must be 2.

%%%%%%%%%%%%% v1.4 main news respect to v1.3 %%%%%%%%%%%%%
02/05/2019
- able to fit profiles with fitting curves (specified by user), plot the fitting and output coeffients in file
- the 1Dplotter is now a function that is called by EnFloSoftwareMatlabSuite
- The format (String or numeric) of the input variable is specified in inputVarDefault.m
- a system of errors has been implemented which does not stop the code but output the msg in a error log file (by means of EnFloSoftwareMatlabSuite)

%%%%%%%%%%%%% v1.5 main news respect to v1.4 %%%%%%%%%%%%%
08/05/2019
- The fitting profile feature has been extended so that it can now accept multiple starting points for the coefficients and select the one with the final smallest error

- The fitting coefficient file is now written on a dedicated function which is able to know the maximum number of coefficient column needed and adjust the headers consequently
- All the array and matrix sizes are now declared in advance to reduce memory fragmentation
- The 4 row space for the headers on top of the data files has been reintroduced
- Two new input variables have been added to correctly place the picture in the single plotter in case PDF format is chosen (they are sLeftEdge and sBottomEdge)

%%%%%%%%%%%%% v1.6 main news respect to v1.5 %%%%%%%%%%%%%
13/05/2019
- Added the double Y axis feature. Now it is possible to choose for each profile if it refers to the left or right Y axis. New input variables have been added (fAxisLabelYright, fAxisLimYright, fYrightAxisColor, pAxisYright) while existing variables have modified functionality (fLogAxis, fTickLabel). In case, at least one profile in the figure refers to the right axis, fLogAxis and fTickLabel must have three values, the last controlling the right Y axis.

%%%%%%%%%%%%% v1.7 main news respect to v1.6 %%%%%%%%%%%%%
15/05/2019
- Added locationPlotter which allows to show the location of the measuring points in a 3D geometry stl. Many geometries can be added, each through a dedicated line in the Input1D.xls file. It is possible to specify the figures whose points will be shown through a dedicated input variable.
- Also a reference point and arrow can be shown in the location plotting

%%%%%%%%%%%%% v1.7a main news respect to v1.7 %%%%%%%%%%%%%
03/09/2019
- Removed check on number of result file columns to add possibility of metadata columns.

2Dplotter

%%%%%%%%%%%%% v1.0 main features %%%%%%%%%%%%%
03/04/2019
- produce 2D contour plot starting from a 3 column result file
- all inputs are given through an input file
- all input variables have preassigned default values in the code, so only the ones that you want to modify have to be specified in the input.xls file
- the image is saved in a file automatically. Multiple formats are available
- it can accept multiple result files and produce a separate plot for each
- an interactive window for each plot opens, which allows some operations

%%%%%%%%%%%%% v1.1 main news respect to v1.0 %%%%%%%%%%%%%
05/04/2019
- implemented a multiplot function to save all the graphs in a single file
- colorbar shown for each graph
- logarithmic scale colormap
- each row in the input.xls file now refer to a different graph

%%%%%%%%%%%%%%%%% v1.2 main news respect to v1.1 %%%%%%%%%%%%%%%%%
09/04/2019
- produce scatter, streamlines and vector graphs
- only one colorbar can be plotted in the multiplot if it is the same for all the graphs
- axis labels in the multiplot are be shown only for the corner axes in the page, in case they are the same for all the graphs
- the order in which the various types of plots are stacked on the figure can be altered on demand by means of the variable bplotOrder. The default order is contour>scatter>streamline>vectors>geometry
- a geometry plotter has been added but it has very limited features.

Note for version 1.2

Regarding the single colorbar in the multiplot, two methods are available. If the number of graphs is lower than the slots available in the page, the colorbar will be plotted in a dedicated empty slot. In case all the slots are already occupied by graphs, only the colorbar for the last graph will be shown. In the latter case, in order to keep the spacing constant among all the graphs, the colorbar for the other graphs will be plotted as well, but not shown.

%%%%%%%%%%%%%%% v1.3 main news respect to v1.2 %%%%%%%%%%%%%%%
25/04/2019
- legend and labels plotted in single figure only if needed, to save time
- generated multiplot file automatically opened
- '/' replaced by '\' in file and folder name
- axis label ticks can now be deactivated for each graph independently

%%%%%%%%%%%%%%% v1.4 main news respect to v1.3 %%%%%%%%%%%%%%%
07/05/2019
- the 2Dplotter is now a function that is called by EnFloSoftwareMatlabSuite
- The format (String or numeric) of the input variable is specified in inputVarDefault2D.m
- a system of errors has been implemented which does not stop the code but output the msg in a error log file (by means of EnFloSoftwareMatlabSuite)
- All the array and matrix sizes are now declared in advance to reduce memory fragmentation
- Two new input variables have been added to correctly place the picture in the single plotter in case PDF format is chosen (they are sLeftEdge and sBottomEdge)
- the single colorbar mode has been improved: not the colorbar is plotted on a superimposed invisible graph whose axis range from 0 to 1. Single colorbar size, location and orientation is manually defined

%%%%%%%%%%%%%%% v1.5 main news respect to v1.4 %%%%%%%%%%%%%%%
11/05/2019
- geometry plotter fully working, able to read multiple 3D geometries from stl files. Extract a 2D plane and plot it in the figure
- user defined colormap loaded from a separated file
- multiple plots superimposed in the same figure by specifing same fFigureNr
- Output a datafile with contour graphs data for each figure

%%%%%%%%%%%%%%% v1.6 main news respect to v1.5 %%%%%%%%%%%%%%%
23/05/2019
- Added locationPlotter which allows to show the location of the measuring points in a 3D geometry stl. Many geometries can be added, each through a dedicated line in the Input2D.xls file. It is possible to specify the figures whose points will be shown through a dedicated input variable.
- Also a reference point and arrow can be shown in the location plotting
- geometry plotter has been improved and it is now able to slice the stl in any plane in 3D. Starting from the location of each measuring point it is now able to make a least square fitting finding the normal to the fitting plane. Such normal is used to compute the new points location relative to the fitting plane reference system. Since the simple indication of the points on a plane does not allow to univocly identify the normal (given the uncertainty on the normal sign) it is possible that the figure may appear rotated respect to the desired view. If this happens the user can act on the sign of the normal directly by changing the value of the variable fFlipNormal. This variable allows to flip either the normal to the fitting plane or the normal to the reference plane (which is always the plane xy with normal [0 0 1]). A proper combination of the two should allow to obtain always the correct picture.
- The automatic computation of the fitting plane is triggered by leaving empty the forth and fifth column in the datafile. In case the user wants, the X and Y axis in the plot can stil be decided manually by filling the two columns(particularly useful in case of non physical axis specification).
- For what concern the slicing plane for the geometry plotter, two scenarios are available: if gGeometryPlanarPoints is left empty, the same plane computed from the sampling point location will be used (note that in case the X Y axis is manually specified, then gGeometryPlanarPoints cannot be left blank); if three points coordinates are indicated in gGeometryPlanarPoints the slicing plane is computed from such points and it is independent from the plane eventually computed for the measuring points. The latter option is particularly useful in case the user wants to use for the geometry slice a plane different from the one in which

the measurement where performed (an example is if the user wants to show in transparency buildings that are not interesected by the fitting plane).

%%%%%%%%%%%%%%% v1.7 main news respect to v1.6 %%%%%%%%%%%%%%%
25/05/2019
- The vectorial quantities (e.g. velocity or fluxes components) have now the capability to be rotated in the new reference system as the measuring point locations. In the input datafiles now everytime the last two or three columns have to be specified for the vectorial quantities. If the third column is left empty (e.g. with or without the header) the quantities specified in the first two columns will not be rotated and will be applied as they are. In case also the third column is filled and the position columns (forth and fifth) are left empty (triggering the plane fitting computation) the vectorial quantities will be rotated in the fitting plane reference system following the same procedure applied for the measuring points location. A new input variable has been introdued: fVectorRotations. It specifies the angles of the acquired vectorial quantities respect to the measuring reference system. As example, if the LDA is align with the tunnel reference system while the model reference system is given as input, an angle has to be provided to rotate the velocity and align it with the model reference system (but remember, any rotation will took place only if all the three components are specified). In case the plane of rotation is parallel e.g. to the floor and only the two horizontal components of the velocity are known, the vertical component can be filled with zeros, in order to allow the rotation, since these zeros values will not been used in any case (being the fitting plane perpendicular to them).

%%%%%%%%%%%%%%% v1.8 main news respect to v1.7 %%%%%%%%%%%%%%%
04/06/2019
- plot2D3D has been introduced which allows to plot with the 3D geometry the contour graphs previously plotted in 2D.
- plot2D3D includes also locationPlotter, and as it was for the latter, it must refer to previous input line (which has to be specified in ldListDatafileShown).
- if the user wants to show in the multiplot only the 3D graph(s) including multiple plots inside, they still need to refere to previous input file lines (each one linked to one datafile). The latter should report the same figure number to skip their plotting.

%%%%%%%%%%%%%%% v1.9 main news respect to v1.8 %%%%%%%%%%%%%%%
01/09/2019
- The flipNormal function does not allow in some cases to approriately show the plot in the desired axis orientation. For this reason a reverseAxis function has been introduced, which allows to reverse either the x or y-axis respect to the previous orientation, by leaving unchanged the axis label in the figure. This is equivalent to flip the normal and see the plane from the opposite view. This funcion has effects also on the contour writing file.

LPplotter

%%%%%%%%%%%%% v1.0 main features %%%%%%%%%%%%%
23/05/2019
- produce 3D geometry from binary or ASCII stl files. Multiple stl file can be loaded in the same figure, shown with e.g. different colors
- plot the measuring points indicated in the datafiles. If the user want to shown points with different properties, they have to be inserted in different datafile, and for each a different line in the input file has to be specified.

Suite

%%%%%%%%%%%%% v1.0 main features %%%%%%%%%%%%%
06/05/2019
- This first version of the suite incorporates 1Dplotter v1.5 and sevenHoleProbe v1.5 (2Dplotter and 3Dplotter have still to be implemented)
- When calling for the suite a string has to be send on TCP. The sintax is two numbers (01 for 1Dplotter and 04 for sevenHoleProbe) followed by a semicolon and blank spaces to fill the string length indicated in the EnFloSoftwareMatlabSuiteSettings.xls file. The output string after the programme run will be of the same

length with the same starting. If an error occured during the call an error message will be send after the semicolon in the string. At the same time an error log file will be written.

%%%%%%%%%%%%% v1.1 main features respect to v1.1 %%%%%%%%%%%%%
07/05/2019
- 2Dplotter v1.4 added to the suite

%%%%%%%%%%%%% v1.2 main features respect to v1.1 %%%%%%%%%%%%%
11/05/2019
- 2Dplotter v1.5 added to the suite

%%%%%%%%%%%%% v1.3 main features respect to v1.2 %%%%%%%%%%%%%
13/05/2019
- 1Dplotter v1.6 added to the suite

%%%%%%%%%%%%% v1.4 main features respect to v1.3 %%%%%%%%%%%%%
23/05/2019
- 1Dplotter v1.7 added to the suite
- 2Dplotter v1.6 added to the suite
- locationPlotterStandAlone v1.0 added to the suite

%%%%%%%%%%%%% v1.5 main features respect to v1.4 %%%%%%%%%%%%%
25/05/2019
- 2Dplotter v1.7 added to the suite

%%%%%%%%%%%%% v1.6 main features respect to v1.5 %%%%%%%%%%%%%
04/06/2019
- 2Dplotter v1.8 added to the suite

%%%%%%%%%%%%% v1.7 main features respect to v1.6 %%%%%%%%%%%%%
01/09/2019
- 2Dplotter v1.9 added to the suite

%%%%%%%%%%%%% v1.7a main features respect to v1.7 %%%%%%%%%%%%%
03/09/2019
- 1Dplotter v1.7a added to the suite