1 Postprocessing Tools

Here you find a list of possible postprocessing tools and how to use them.

1.1 GiD

This chapter describes some relevant aspects of the postprocessing step and the way to load results from a numerical analysis into GiD. In the GiD postprocess you can study the results obtained from a solver program. The GiD postprocess receives mesh and results information from the solver module. If the solver module does not create any new mesh, the preprocess mesh is used.

The solver program has to write the results in a file that have the extension *.post.res* and its name must be the project name. If the solver writes a mesh, the file must have the extension *.post.msh*.

To perform postprocessing with GiD: Start GiD and directly select the postprocess option from the *View* menu. Now open the files *ProjectName.post.res* and *Project-Name.post.msh*.
Then all the geometry and mesh information as well as the result of calculation will be read by GiD.

Inside the postprocessing section of GiD, all the visualization features and management options of the preprocess section like: Zoom, Rotate (Rotate screen/object axes, Rotate trackball, etc.), Pan, Redraw, Render, Label, Clip Planes, Perspective, ... are available.

The possible results to be displayed can be grouped into five major categories:

- **Scalar view results:** Show Minimum & Maximum, Contour Fill, Contour Text Ranges, Contour Lines, Iso Surface, and its configuration options.
- **Vector view results:** Mesh deformation, Display Vectors, Stream Lines (Particle Tracing)
- **Line diagrams:** Scalar line diagram and vector diagrams.
- **Graph lines:** XY plots.
- **Animation:** Animation of the current result visualization.

The *View results* menu and the *View results window* (see figure 1.1) are used to manage the visualization of the different type of results. The *View results* menu lets you select

any type of results such as *Contour fill, Contour ranges, Contour lines, Diplay vectors, Show min/max, Iso surfaces, Stream lines, Graphs, Deformation* and *Line diagrams*. The *View results window* manages the visualization of these type of results: *Contour fill, Contour ranges, Contour lines, Diplay vectors, Show min/max, Scalar, and Vector line diagram*.

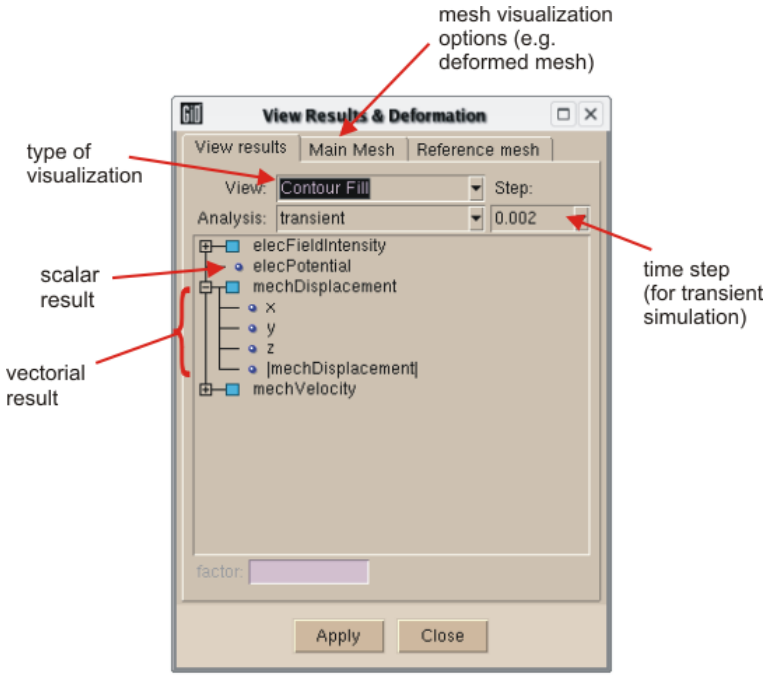

Figure 1.1: *View result window*

1.1.1 Show Minimum and Maximum

View results → Show Min Max

With this option the user can see the minimum and maximum of the chosen result.

1.1.2 Contour Fill

View results → Contour Fill

This option allows visualization of the colored zones, in which a variable or a component, varies between two defined values. GiD can use as many colors as allowed by the graphical capabilities of the computer. When a large number of colors is used, the

variation of these colors looks continuous, but the visualization becomes slower. (see figure 1.2)

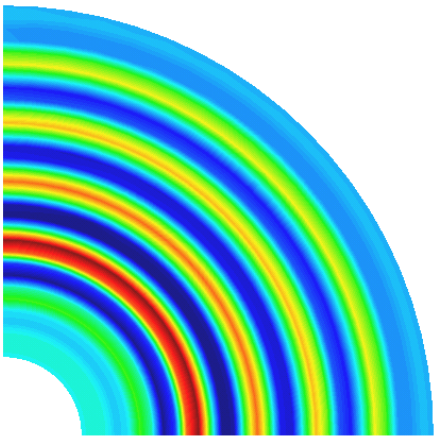

Figure 1.2: *Example for contour fill*

Several configuration options can be accessed via the **Options** → **Contour** menu.

- **Number Of Colors:** Here the number of colors can be specified.
- **Set Limits:** This option is used to tell the program which contour limits should be used when there are no user defined limits.
- **Limits:** The absolute minimum and maximum of all sets or shown sets, for the actual step or for all steps can be specified.
- **Max/Min Options:** Inside this group, options for the minimum or maximum value can be defined:
  - *ResetValue:* User can reset the maximum/minimum value, so that the default value is used.
  - *OutMaxColor / OutMinColor:* With this option user can specify the way that outlier values are drawn: Black, Max/Min Color, Transparent, or Material.
  - *Def. MaxColor / Def. MinColor:* This option lets the user define colors for displaying the minimum or maximum values.
- **Color window:** A window is popped up to let the user configure color scale for the contours. (see figure 1.3)

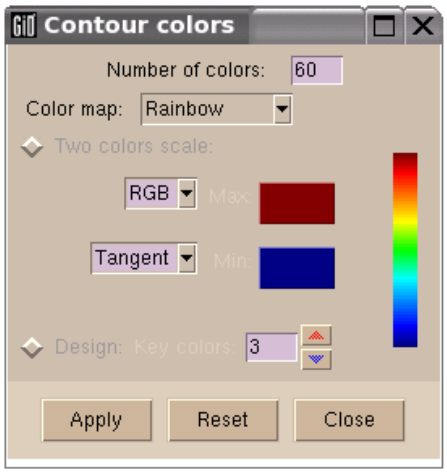

Figure 1.3: *Contour color window*

1.1.3 Contour Lines

View results → Contour Lines

This display option is quite similar to *Contour Fill*, but the isolines of a certain nodal variable are drawn. In this case, each color connects all points with the same value of the variable chosen. The configurations options are almost the same as those for *Contour Fill*. The only difference is that the number given in the *Number of Colors* option will be used as the number of lines for this contour lines representation. For an example see figure 1.4.

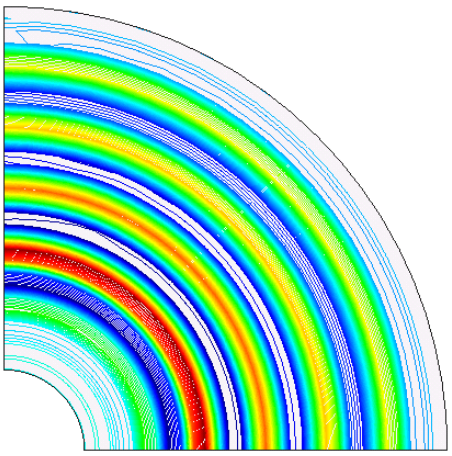

Figure 1.4: *Example for contour lines*

1.1.4 Display Vectors

View results → Display Vectors

A menu is displayed for selection purpose. The program will then show the nodal vectors of the selected result. These vectors can be interactively scaled and factors can be applied as a number of occasions. The color of the vectors can also be differed by the user. The default color is green.

1.1.5 Iso Surfaces

View results → Iso-Surfaces

A surface is drawn that connects all points with the same value inside a volume. There are several options for creation Iso-Surfaces:

- **Exact:** After choosing a result or a result component of the analysis step, the user can input several fixed values. Then for each given value an Iso-Surface is drawn.
- **Automatic:** Similarly, after choosing a result or a result component, the user is asked for the number of Iso-Surfaces to be created. GiD calculates the values between the minimum and maximum values.
- **Automatic Width:** After choosing a result or result component, the user is asked for a width. This value is used to create as many Iso-Surfaces as needed between minimum and maximum values.

Several configuration options can be accessed via the *Options* menu:

- **Display Style:** There is also a *Display Style* option similar to Volumes/Surfaces/Cuts for Iso-Surfaces. The options *All Lines, Hidden Lines, Body, Body Lines* are also avaliable.
- **Render:** This option renders the Iso-Surfaces to *Flat* or *Normal*.
- **Transparency:** The Iso-Surfaces can be set to *Transparent* or *Opaque*.

1.1.6 Deformation

View results → **Deformation**
or
Window → **View Results** → **Main Mesh**

Vectorial results can be mapped as offsets to the nodal coordinates, so the mesh gets deformed. This is especially useful for visualizing mechanical displacements.

The following options are possible (only in **View Results** → **Main Mesh**-window, ref. fig. 1.5):

- **Original** / **Deformed**: Select if the original (undeformed) or the deformed mesh should be drawn.
- **Step**: The deformations of a particular time step can be chosen (only transient simulation).
- **Result**: List of available vector results (e.g. mechDisplacement, mechVelocity), which are mapped as deformation.
- **Factor**: In order to visualize very small deformation values, a suitable factor can be chosen, by which the results are multiplied.

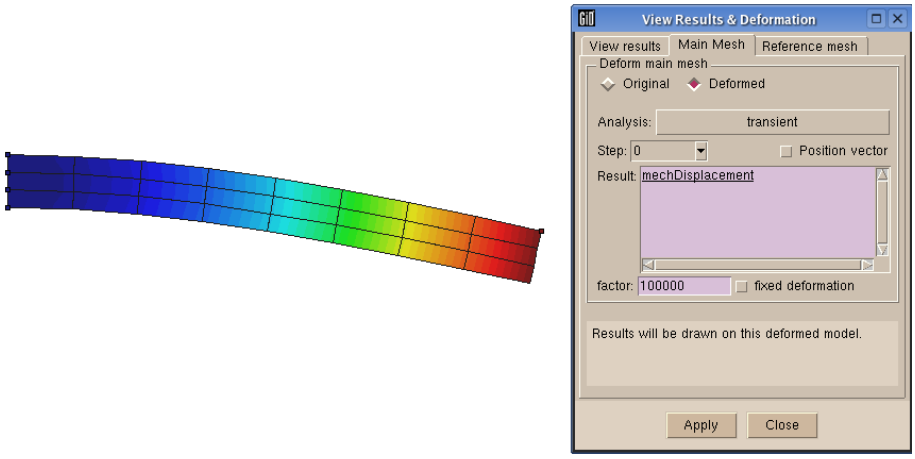

Figure 1.5: *Deformed mesh with additional contour fill*

Note: If also an additional scalar or vector visualization is chosen, this is mapped onto the deformed mesh as well. Therefore it is possible to show for example the acoustic potential as contour fill on a (mechanically deformed) mesh.

1.1.7 Graphs

View results → Graphs

Here the user can draw graphs in order to have a closer look to the results. Several graph types are available: point analysis vs. time, result 1 vs. result 2 over points, and results along a boundary line. Under the *Graphs* option of the *View results* menu there are following options:

- **Show:** Here the user can switch between the graphs view and the postprocessing view.
- **ClearGraphs:** To reset all the graphs and do a new start.
- **Point Analysis:** This graph shows the evolution of a result on a point along all the steps in the current analysis.
- **Point Graph:** After choosing one point or points, the user can contrast a result against another.
- **Border Graph:** After selecting a border, the user can see how the results varies along this boundary.

The user can also view the labels of the points of the graphs, not only the graph points number, but also their X and Y values. When turning into the normal results view, if some points of the graphs are labeled, the labels also appear on the results view. An example graph is shown in figure 1.6.

1.1.8 Animation

Window → Animate

With this window a little part of automatization has been done to create animations inside GiD. This window lets the user create an animation of the current *Results View*, where the limits can be fixed along the animation with *Automatic Limits*, and/or of the *Deformation* of the mesh.

Some useful options are:

- **Automatic Limits:** GiD searches for the minimum and maximum values of the results along all the steps of the analysis and uses them to draw the results view through all the steps.

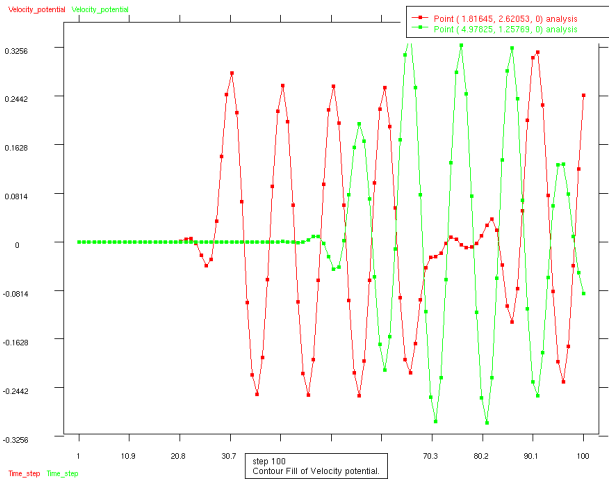

Figure 1.6: Example of graph

- **Delay:** Here the user can specify a delay time between steps in milliseconds.
- **Save TIFF/GIF on:** With this option user can save snapshots of each step in TIFF or GIF format, while the 'Play' button is pushed.
- **Static analysis animation profile:** If the project has only one step, it is possible to simulate an animation by generating the intermediate steps. Use this option to automatically generate several frames, following a linear, cosine, triangular or sinusoidal interpolation between the normal state and the deformed state.

1.2 MATLAB

The commercial software MATLAB is a general purpose mathematic programming environment, which can be used for performing numerical calculations and visualizing results. In the exercise, you can use it for reading in text files (e.g. history files), perform calculations on the results and create graphs.

A list of additional tutorials is available at
`http://www.mathworks.com/academia/student_center/tutorials/`.

1.2.1 Getting Started

To start matlab, just type

```
matlab
```

in the console. The start screen can be seen in Fig. 1.7. When using matlab interac-

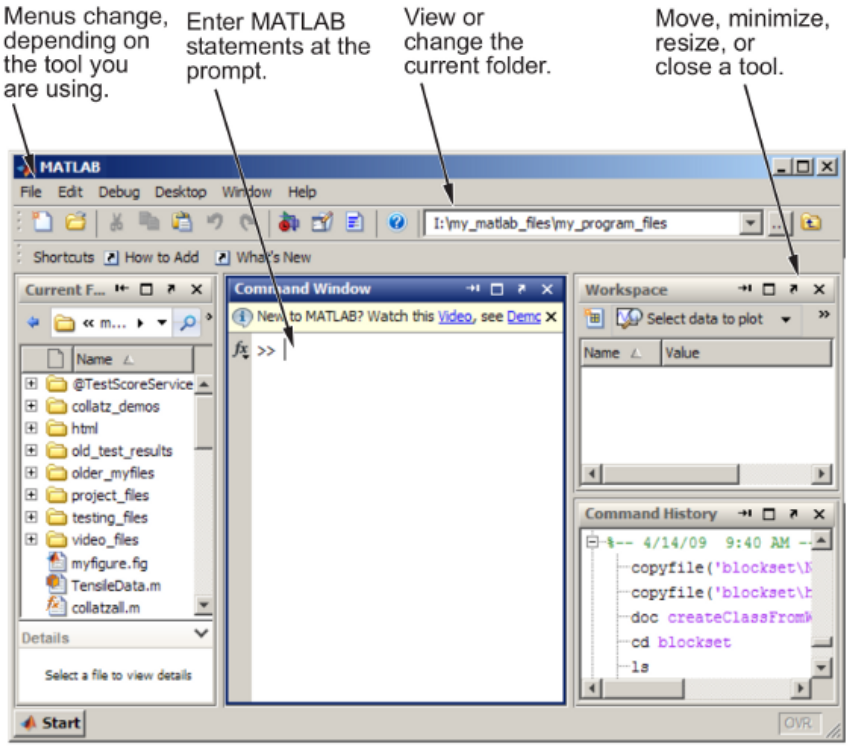

Figure 1.7: Screenshot of MATLAB

tively, the commands get entered in the *Command Window*. MATLAB can be used as a simple calculator:

```
>> 2 + 3/4*5
ans =
5.7500
>>
```

The result of the calculation is implicitly assigned to the variable `ans`. To assign it to a user defined variable, just write

```
>> a = 2 + 3/4*5;
>> a+3
```

```
ans =
8.7500
>>
```

The additional semicolon ';' at the end of the line prevents the echo of the output. To reference a variable, just use its name without any prefix.

Variables must not start with a digit and can not contain any of the characters %, - or @. Upper- and lowercase gets distinguished.

There are many pre-defined functions, e.g. the trigonometric functions `sin`, `cos` and `tan`. To get additional help to a command, just type

```
>> help sin
   SIN    Sine of argument in radians.
   SIN(X) is the sine of the elements of X.

   See also asin, sind.

   Reference page in Help browser
      doc sin
>>
```

Numbers can be entered in various formats:

Type	Example
Integer	123, -456
Real	1.23, -4.56
Complex	1.23 + 4.56 i ($i = \sqrt{-1}$)
Inf	Infinity (division by 0)
NaN	Not a number (0/0)

The internal precision of real numbers is double, i.e. numbers are stored with a precision up to 15 significant digits.

Instead of entering command in the interactive command window, MATLAB can also execute script files, which are just text files, containing the MATLAB commands line by line. The files have typically the suffix `.m`. To execute a script file, just make sure that MATLAB is called from the same directory where the file is located at and simply enter the name of the script file without the extension `.m`:

1.2.2 Vectors / Matrices

A row-vector is simply defined by the help of square brackets

```
>> v = [4, 5, 6, 7];
>> size(v)
ans =
   1     4
```

The result of the `size` command tells us, that v is a 1 x 4 matrix, i.e. a column vector.

To access individual entries of a vector, use the ()-operator, which uses **1-based** indexing:

```
>> v(2)
ans =
   5
```

To access a range of a vector, we can use the ':'-operator:

```
>> v(2:4)
ans =
   5     6     7
```

We can apply standard operations on vector, like transpose ('-operator) or calculation of the norm:

```
>> v'
ans =
    4
    5
    6
    7

>> size(v')
ans =
   4  1

>> norm(v)
ans = 11.2250
```

```
>> sqrt(v)
ans =
   2.0000    2.2361    2.4495    2.6458
```

Note, that internal functions like `sqrt` can be directly applied to vectors and return again a vector with the element-wise result.

For some operations, like e.g. the exponential operator `^` we have to explicitly tell MATLAB, to apply the operation element-wise using the postfix .-operator:

```
>> v^2  % MATLAB would try to calculate v * v, i.e. vector-vector product
??? Error using ==> mpower
Matrix must be square.

>> v.^2 % element-wise application of the ^-operator
ans =
   16    25    36    49
```

To produce a vector containing linearly spaced values, the command linspace can be used:

```
>> x = linspace(0,2*pi,100)
x =
 Columns 1 through 7
 0    0.0635    0.1269    0.1 ...
 ...
 Columns 99 through 100
 6.2197    6.2832
```

This will produce a column vector of size 100 with a range from 0 to $2 * \pi$.

To define a general matrix, we can use the following syntax

```
>> m = [1,2,3; 4,5,6; 7,8,9]
m =
   1     2     3
   4     5     6
   7     8     9
```

Here the semicolon ';' denotes the end of a row. We can also use here the colon notation (:) to address a sub-matrix

```
>> m(2:3, 1:2)
ans =
     4     5
     7     8

>> m(1,:)
ans =
     1     2     3
```

1.2.3 Generating Graphs

A linear plot of a one-dimensional function is simply generated by the `plot` command

```
>> x = linspace(0, 2*pi, 100);
>> y = sin(x).^2
>> plot(x,y)
```

The resulting plot can be seen in Fig. 1.8. We can add additionally a title for the graph, as well as for the x- and y-axis. In addition, we want to activate the background grid and save the plot into the file `graph.png`:

```
>> title('My first plot')    % set figure title
>> xlabel('time (s)')        % set x-axis label
>> ylabel('amplitude (m)')   % set y-axis label
>> grid()                    % enable grid
>> saveas(gcf,'graph.png')   % save graph into file
```

The command `gcf` in the last row returns a so-called *handle* to the global figure, which gets saved to the file `graph.png`.

To produce several graphs in on figure, we have to issue the `hold()` command, followed by several `plot()` commands for the different graphs (see Fig. 1.9):

```
>> x = linspace(0, 2*pi, 100);
>> plot(x,sin(x).^2,'r-o')         % specify line style red, dashed line
                                   % with o-markers
>> hold()                          % force MATLAB to keep the figure
>> plot(x,cos(x).^2,'b-x')         % specify line style blue, dashed line
                                   % with x-markers
>> legend('sin(x)^2', 'cos(x)^2')  % add legend for both plots
```

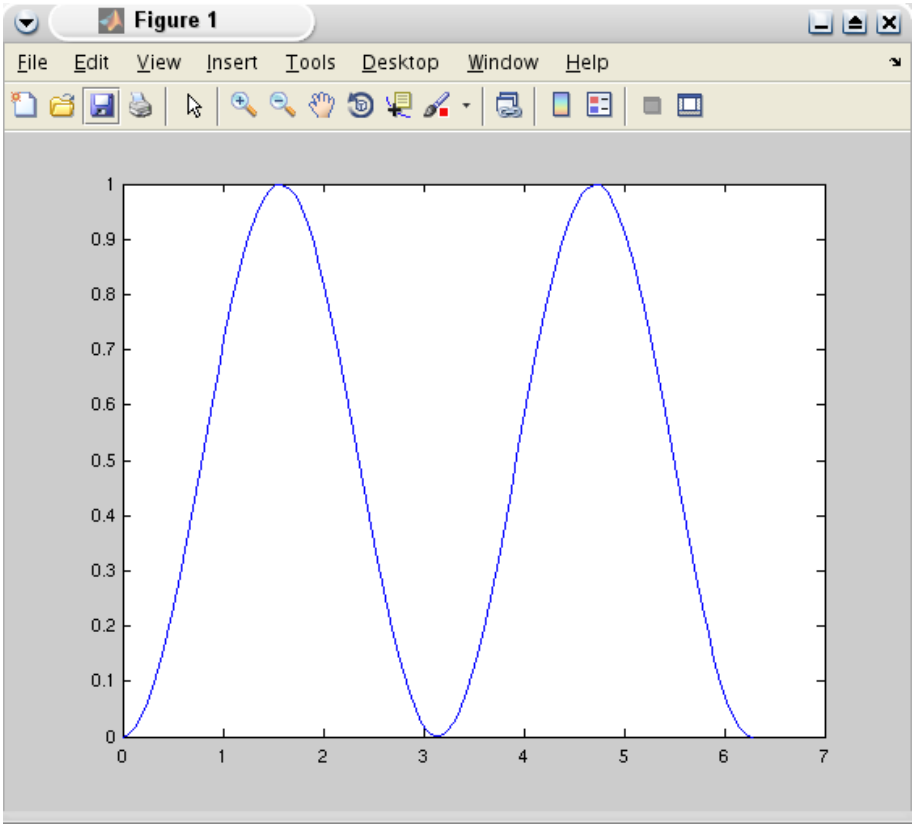

Figure 1.8: Linear plot of $sin(x)^2$

```
>> title('Multi plot')
>> xlabel('time (s)')
>> ylabel('amplitude (m)')
>> grid()
>> saveas(gcf,'graph.png')
```

The third argument to the `plot()` command is a style string, which is composed as follows:

- The first character specifies the color of the line.
- The other characters specify the line style (type of line, marker symbols).

Here is a short list of available colors and line/marker styles:

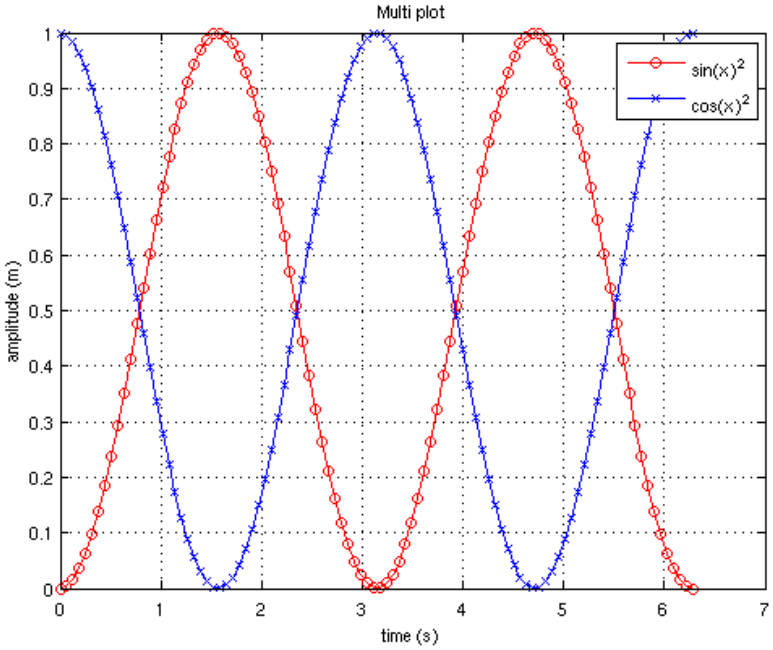

Figure 1.9: Several plots in one figure

Colors		Style	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		–	dashed

For a complete list of available colors / styles, use

```
>> help plot
```

There are several other plot commands available:

- `loglog()`: Generates a double-logarithmic plot.
- `semilogx()`: Generates a plot with logarithmic x- and linear y-axis.

- `semilogy()`: Generates a plot with linear x- and logarithmic y-axis.
- `polar()`: Generates a plot in a 2D polar coordinate system (e.g. for acoustic radiation pattern).

Additional information about the plotting facilities of MATLAB can be found here:
`http://www.mathworks.com/help/techdoc/learn_matlab/f3-8955.html`

1.2.4 Reading / Writing Text Files

To visualize the results of a single node / element of a CFS simulation run, we have to read in space-delimited text files. Here is an example of a history file:

```
# Result: 'acouPressure' on node(s) 'observ-0' number #1
# t (s)   (Pa)
# ------------------------------------------------------------------------
8.33333e-09  5.30399e-05
1.66667e-08  8.98755e-05
2.5e-08  0.000134436
3.33333e-08  0.000186659
...
```

Such a file can be loaded using the `importdata()` function:

```
>> a = importdata('file.txt');
   a =
     data: [9600x2 double]          % contains the numerical values
     textdata: {3x2 cell}           % containts the header text
     colheaders: {'#'  [1x75 char]} % contains the column headers
```

The result variable `a` is a so-called struct (similar as in C / C++), which contains several variables. The numerical values are stored in the member variable `data`:

```
>> plot( a.data )
```

To write out an array in a plain text file, one can use the command `save` with the additional option `-ASCII`

```
>> b = [1, 2, 3, 4; 0.1, 0.2, 0.2 0.4]';
>> save 'out.txt' b -ASCII
```

1.3 OCTAVE

OCTAVE is an open-source clone of MATLAB and is freely available. Most of the commands and the syntax are the same as in matlab. However, it does not have a graphical user interface. It uses internally Gnuplot (see next section) to produce graphical output.

To start octave, just type

```
octave
```

in the console.

To produce a plot of the data containted in the file `signal.txt`, we can use the following script:

```
% Octave can handle doulbe quotes "". In addition the load() command
% is able to treat the header correctly (in contrast to MATLAB)
a = load("signal.txt")

title('My first plot')
xtitle('time (s)')
ytitle('amplitude (m))
grid()
plot(a(:,1), a(:,2), 'r-o')

% Use the print() command to export a graph
print('out.png')
```

For a list of differences between matlab and octave, have a look at the following sites:

- `http://www.gnu.org/software/octave/FAQ.html#MATLAB-compatibility`
- `http://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MA`

1.4 Gnuplot

Gnuplot is a lightweight graph generating and calculation program.

To start Gnuplot, just type

```
gnuplot
```

in the console.

To produce a simple plot of a two-column text file `signal.txt`, we can use the following script:

```
set title "My first gnuplot plot"
set xlabel "time (s)"
set ylabel "amplitude (m)"
set terminal png
set grid
set output "out.png"
plot "signal.txt" using 1:2 with linespoints
```

The resulting graph can be seen in Fig. 1.10 The `set terminal` statement tells gnuplot

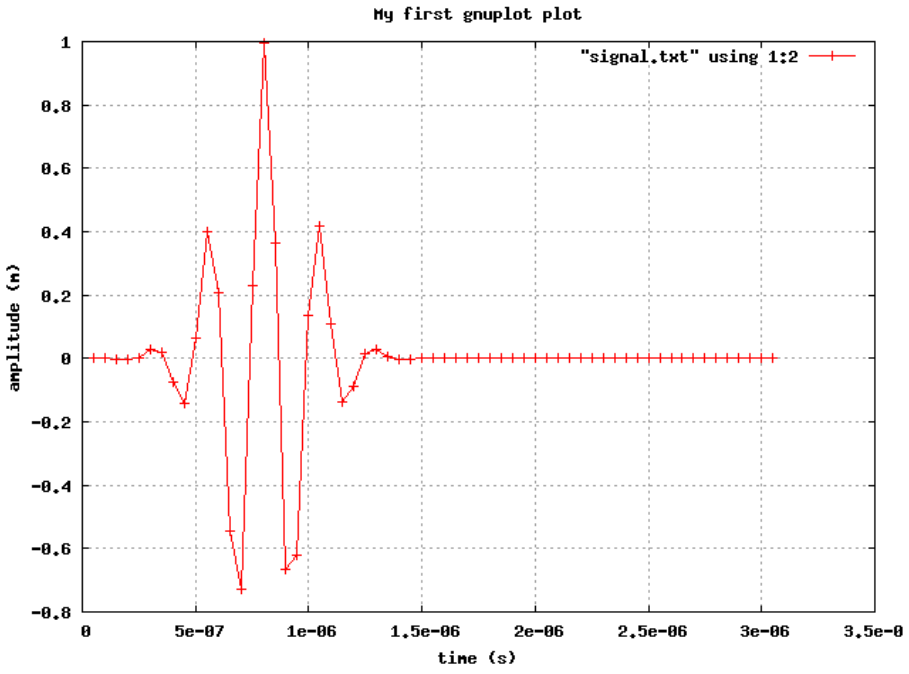

Figure 1.10: Resulting graph

to use the pseudo `png` terminal as output, i.e. it should generate a file in png format. By default, the graph is just written to the screen. The last command `plot` takes directly the filename as argument, so we do not have to load the content explicitly. The option

`using` denotes the columns to be used as x- and y-axis and the last argument specifies the style of the plot.

A very good tutorial for Gnuplot can be found here: `http://t16web.lanl.gov/Kawano/gnuplot/inde`