The Graphics Program *cnom2.0*
User Manual

Susanne Krömker

Interdisziplinäres Zentrum
für Wissenschaftliches Rechnen der
Universität Heidelberg
Im Neuenheimer Feld 368
D-69120 Heidelberg

March 4, 1997

Contents

1 Introduction

1.1 Technical information

cnom2.0 is a visualization software to run on all SGI machines (Reality engine, Indigo, Personal Iris, Indy) and on other machines, e.g. IBM workstations (with graphic adapter) on the basis of the SGI-GL graphics library. An *OpenGL* version is in progress.

1.2 Description

Three dimensional data e.g. orbit data in a phase space or solutions of vector valued differential equations over a two dimensional regular or irregular grid can be displayed and transformed with the program. Several such function graphs (at maximum 100) over the same grid can be hold in the same data file. They can be shown in one or two adjacent coordinate systems, mapping another function value as color information onto the surface(s). At maximum three sets of z-data can be compared at a moment. Furthermore one can keep track of a function through different files (time steps) while the grid may be changing. The program is menu and mouse driven, but for some commands also a shell based dialogue is required. An animation part enables the user to write and replay a sequence of previously stored scenes, including transformation routines, time series, varying function values or data files.

Figure 1: A spiral of a reaction diffusion system visualized with *cnom2.0*.

1.3 Contact

The binaries and examples are available at the anonymous ftp-server with the address **ftp.iwr.uni-heidelberg.de** in the directory /pub/cnom. In case you have any questions or any helpful suggestions on what should be improved, please contact:

Susanne Krömker

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen der
Universität Heidelberg
Im Neuenheimer Feld 368
D-69120 Heidelberg

Phone: +49-6221-54-8883 or +49-6221-54-4949 (Graphics lab)
e-mail: susanne.kroemker@iwr.uni-heidelberg.de

2 Getting started

2.1 Installation

Ask the system administrator to install the executable file at a location on the machine accessible to all users of the program (e.g. in /usr/local/bin/) or install it in your home directory, and name it shortly *cnom*. Make sure to have changed the permissions mode by typing for example 'chmod 755 cnom' in order to make it executable. Include the location of the program into the search path of your favorite shell.
To get an impression of the program, it might be helpful to install as well the examples directory following the README file. For the different examples in the subdirectories further information is provided in form of README files.
There is also a ⟨*script*⟩ called 'demoscript', that can be run following the instructions in the README file. Then you will get a survey of the possibilities of the program.

2.2 Interaction of program, input file and data files

The program, the data files and a data specific ⟨*input file*⟩ belong together. The role of the ⟨*input file*⟩ is to introduce the specific data structure to the program and to restore certain settings once chosen during previous executions of *cnom2.0* (see 3 and 5.1 below). Keep the ⟨*input file*⟩ in the same directory as the data, otherwise it will not be found, and the program will not start. Upcoming user dialogues provide the program with the remaining information needed to display the data.

The data file names consist of a base name and a number as a suffix, with a leading 0 for those numbers less than ten (just to receive a clear listing in the data directory). An extension '.bin' indicates a binary format, which will be used preferentially (see 4.7 below).

2.3 How to start the program

Change the current working directory to the one containing the ⟨*input file*⟩ and data files. Then simply type *cnom*, and a shell based dialogue will ask for the base name of the data files and the last counter of the sequence. The default name for the ⟨*input file*⟩ is 'input', the default name for the ⟨*script*⟩ is 'script'.
In case you want to supply other input than the default you start the program with:

cnom [-option ⟨*name*⟩]

where the following command line options are available:

-i ⟨*input file*⟩
-s ⟨*script*⟩
-d ⟨*base name of data files*⟩
-c ⟨*last counter of data files*⟩

For example:

cnom -i my_input -s my_script -d my_data -c 20

will start *cnom2.0*, read the file my_input, load the file my_script and display data of the file my_data01 scaled with the extremal values over the first 20 files with the same base name, my_data02, my_data03, etc.
A wrong usage of these options will result in a help screen as follows:

Usage: *cnom* [-i ⟨inputfile⟩] [-s ⟨scriptfile⟩] [-d ⟨data⟩] [-c ⟨counter⟩] Default names are 'input' for ⟨*input file*⟩ and 'script' for ⟨*script*⟩

A dialogue leads through the next steps, asking for information which was not supplied in the command line. It expects the base name of the data now, and the maximum number of the sequence of files you want to look at (counter).
Then you are asked for one of the shortenings of a list of possible z-data sets to display the first object. If you put a dash in front of the shortening, scaling

of this data set is done according to the maximal amplitude (maximum minus minimum) detected within the set of data files. This relative scaling is useful when minimal and maximal values differ only slightly and you want to blow up this difference.
At last you are asked for one of the z-data sets to be mapped onto the surface as color shading.

2.4 A brief guide to the program

A window in the lower left part of the screen displays the data, and another window at the bottom of the screen displays information about the data. Press the RIGHTmouse button button while the cursor is inside one of those windows belonging to the program. The pop-up menu appears. Select a command of the menu or some submenu and press the RIGHTmouse button button again. In this description the pop-up menu commands are printed in *slanted style*. They are explained in detail in 5.1 below. If you invoke a transformation routine (translation, rotation or zooming), and a blue bar appears in the bottom window, the x-screen coordinate of the mouse pointer (anywhere on the screen) will be used to transform the data according to the values of the scale at the bar, while the RIGHTmouse button is pressed. Try the different views under the command *Restore views* and see what happens. If there exists already a ⟨*script*⟩, try the menu command *Dialogue* ▷ *Read* ⟨*script*⟩, and within a dialogue in the upcoming shell you will be asked for the first and last scene you want to look at.
The settings achieved during runtime can be saved to the ⟨*input file*⟩ only if exiting by the menu commands *EXIT* ▷ *Save all*, *EXIT* ▷ *Save views* and *EXIT* ▷ *Save colors*. If exiting by *EXIT* ▷ *Escape* or pressing the keyboard key ⟨ESC⟩, the ⟨*input file*⟩ remains unchanged.

3 Description of the input file

The program will not start, when there is no ⟨*input file*⟩ with the default name 'input' or no file with the name specified in the command line with the option flag -i. The ⟨*input file*⟩ contains all the necessary information in order to interpret the data from the data file correctly.
The ⟨*input file*⟩ consists of a set of keywords (in the following shortly called items) and assigned values. The items all have to be written in capital letters followed by an equal sign. The order of the items is not essential, with a few exceptions. Comments begin with a # sign.
Here a listing with explanations of all possible items of the ⟨*input file*⟩ is given.

The essential items (see 3.1 below) have to be supplied, resulting in an ERROR message and termination of the program if they are missing. All other items of the ⟨*input file*⟩ have default values, for example those beginning with the letters `STD...` (standard) have a default value of 1. These values may be changed by direct editing in the ⟨*input file*⟩ or automatically when exiting the program and saving the adjustings you made during runtime. Therefore the program distinguishes between savable items (see 3.2 below) and editable items (see 3.3 below). Editable items can only be changed when directly edited in the ⟨*input file*⟩. Warnings will be given for missing items or unexpected strings. To suppress these warnings, supply a line `SUPPRESS = 1` directly as the first item line. This item can only be edited directly in the ⟨*input file*⟩.

`SUPPRESS =` [INTEGER]
0 will not suppress warnings (default),
1 will suppress all warnings.

3.1 Essential items

Essential items are those belonging to the data structure. There is no way of giving them a default value; this is the case for the items `GRID`, `CHOSE` (or `CHOSE`n), and `NNNN`. The program checks for these items first and will terminate if they are missing.

3.1.1 Types of grid

The grid type value indicates what type of grid will be provided. Negative numbers are reserved for data not represented over x-y-grid such as orbits, even numbers less than 10 are reserved for Cartesian grid (lattice grid, strictly rectangular, not necessary equidistant), numbers greater than 10 are reserved for unstructured grid.
One of the following values can be used up to now:

`GRID =` [INTEGER]
-1 orbital data,
1 data on a structured grid
2 Cartesian grid, unchanged x:y proportion,
4 Cartesian grid scaling the maximal x-value to unit length,
6 Cartesian grid scaling the maximal y-value to unit length,
8 Cartesian grid scaling both maximal x- and y-value to unit length,
11 unstructured data, only triangles,
12 unstructured data, mixed triangles and rectangles.

3.1.2 Statements handling z-data

In order to enable the program to distinguish between different sets of z-data (e.g. vector valued solutions over the same two dimensional grid) a flexible handling with string identifiers occurring in the data files is necessary. For each set of z-data you have to supply a short expression used in the shell based dialogues to select this set of z-data, and the exact string identifier that marks the begin of this set of z-data in the data file. These two strings are provided on two adjacent lines of the following structure: a line with the item `CHOSE`, an equal sign and an entry (for the dialogue), followed by a line with the item `NNNN`, an equal sign and an entry (same as in the data file). If `CHOSE` is also followed by a line with the item `ZAXIS`, an equal sign and an entry, this entry (string) will be used naming the z-axis in the graphical representation (optional).
In the following these two (or three) lines will be called a `CHOSE` statement. For convenient use you may put a number after each item `CHOSE` and before the equal sign (`CHOSE`*n*). The program does not check this number, it is optional. At minimum a single `CHOSE` statement has to be supplied in the ⟨*input file*⟩. Otherwise the program detects an error and exits. A maximum of 100 different `CHOSE` statements can be handled by the program, using the string supplied after each `CHOSE` in the dialogues to point to different sets of z-data in each data file.

`CHOSE`*n* `=` [CHAR STRING]
Define a single character or short expression to be used within shell based dialogues with the user to select the z-data set specified by the now following identifier provided after the item `NNNN`.

`NNNN =` [CHAR STRING]
Place here the unique identifier that also marks the begin of a set of z-data in the data files. It will be recognized by the program whenever the expression of the previously given line `CHOSE` is referenced in dialogues.

`ZAXIS =` [CHAR STRING]
This text will be displayed at the z-axis, or below the color bar. This is not an essential item. If it is not provided, 'z' is written to the axis.

3.1.3 Minimal input file

The minimal ⟨*input file*⟩ leads to a standard representation of data i.e. the numerical data is displayed in a PALnorm sized window with a standard perspective showing the first quadrant with axes of unit length and a wire frame representation of the surface in standard color and a grey grid.

You can run the program with a minimal ⟨*input file*⟩. For the necessary data files and their structure see 4 below. In the dialogue, all strings you entered after a `CHOSE` item will be listed. Choose one for the object to be displayed, and (possibly the same) for the color shading.
No overall scaling is done. So you might just look at the origin of a coordinate system if your x- and y-data are too large or too tiny. If this is the case, exit the program and add the editable item `FACTOR` (see 3.3 below) to your ⟨*input file*⟩. Adjust it to the order of magnitude of your grid. You might also be able to transform the perspective by zooming in or out but you loose accuracy and will never have a standard view on your data.
The minimal ⟨*input file*⟩ with suppressed warning messages may look like:

```
# begin of input
SUPPRESS  =  1
# list of variables
CHOSE     =  p      # CHOSE is for dialogue
NNNN      =  P(PA)  # NNNN=string identifier in data file
# end of list of variables
GRID      =  2      # 1=structured, 2=Cartesian
# end of input
```

Omitting all comments and the `SUPPRESS` line this reduces to three lines that have to be provided as minimal version of an ⟨*input file*⟩:

```
CHOSE  =  p
NNNN   =  P(PA)
GRID   =  2
```

3.2 Savable items

Here is a description of those items of the ⟨*input file*⟩ which may be changed during execution and saved to the ⟨*input file*⟩ when exiting with the commands *EXIT* ▷ *Save all*, *EXIT* ▷ *Save views* or *EXIT* ▷ *Save colors*. Saving means that new lines of the necessary structure are created in the ⟨*input file*⟩ if the savable item value differs from its default. Items not listed in this section can only be changed by direct editing of the ⟨*input file*⟩, see also 3.3 below.

Symmetry or periodicity

`HALFX =` [INTEGER]
`HALFY =` [INTEGER]

1 does not display symmetry of data along x-/y-axis (default),
2 displays symmetry by reflecting the data at a coordinate axes,
-2 displays data periodically repeated.

For example if `HALFX = 2` (resp. -2) the data provided for the first quadrant of the coordinate system will be reflected at the y-axis (resp. repeated periodically for negative x-data). If the data does not reflect at the axes, this feature is of no use, but a reflection or periodical repetition can be forced anyway.

Thickness of lines

`WIDTH =` $[1, \ldots, 9]$
Set the thickness of lines (in pixels) for data and grid. The default value is 2 since it is the convenient thickness for printer output. The thickness of the axes is not affected by this statement.

Antialiasing

`ANTI =` [INTEGER]

0 switches the antialiasing mode off (default),
1 switches the antialiasing mode on, resulting in a smoothing of lines by tuning the line color with the background color.

Put the antialiasing mode off if it affects the performance of the transformation routines or if the lines are too close together or too thick, such that it is no improvement at all.

Coordinate axes

`STDAXIS =` [INTEGER]

-1 additional cubic outlines of unit length are made,
0 no axes are drawn,
1 sets standard axis representation (default).

Providing additional information

`STDTEXT =` [INTEGER]
The character string entries provided behind the `TEXT`n items (see 3.3 below) are displayed according to this flag.

0 does not write text but a large color bar and large font is used,
1 writes the texts into the explanation window (default).

Style of data display

`HOW = `[$\pm$ INTEGER]
If the value for `HOW` is positive, the data and the underlying grid (or net) are drawn, otherwise no grid (or net) is displayed. Grid (or net) always means grey colored lines connecting the x- and y-coordinates at the base level of z-data. The absolute values indicate the type of interpolation technique to be used. Notice that not every technique is possible for each grid type. Failure will be announced in the explanation window by a warning message. Possible values are:

1 Bezier splines,
2 Cardinal splines,
3 B-splines,
4 Gouraud-shaded polygons (default),
5 Gouraud-shaded patches,
6 black or white polygons, depending on the background color,
7 black or white B-splines, depending on the background color,
8 Non Uniform Rational B-Splines (NURBS).

Different window sizes for video tape recording

`STDWIN = `[INTEGER]
For on-line recording of videos the different video norms (PAL/NTSC) require different sizes of graphics windows. Via menu command it is possible to toggle between a small (approximately a quarter of the full screen) window and a full screen display (without disturbing border). No toggling between PAL and NTSCnorm sized windows is implemented, so once you want to display a NTSCnorm sized window you have to edit the STDWIN item in the ⟨*input file*⟩ and change its value. Then the menu command toggles between this window and the full screen display.

0 yields a full screen display,
1 brings up a graphics window of PALnorm size: 768×576 (default),
2 brings up a graphics window of NTSCnorm size: 646×486.

Handling of viewing matrices

Certain perspective views onto the displayed object can be specified by $4{\times}4$ matrices defining view angles. In the following these matrices are called viewing matrices. A standard viewing matrix is always provided by the program and copied to all the other viewing matrices as defaults. Whatever was stored in the viewing matrices during the execution of the program is written to the ⟨*input file*⟩ when exiting the program by *EXIT* ▷ *Save views*, in this way:

`SPECIAL1 =`
[4×4×REAL]
`SPECIAL2 =`
[4×4×REAL]
`LAST =`
[4×4×REAL]
The viewing matrices `SPECIAL1` and `SPECIAL2` can be used to store viewing perspectives and are loaded to the program if they are found in the ⟨*input file*⟩. They can be restored by using the according menu command. The viewing matrix `LAST` saves the last viewing perspective when quitting the program. It will only be considered for restart, if the item `STDOSPC` is not 1, 2 or 3 (see 3.3 below).

Handling of colors

It is possible to specify RGBmode values for the red, green and blue part of the background color `MAINCOL`, the text color `TEXTCOL` and each of the color fields of the color bar of the explanation window (`C1` up to `C15`). This can be done with a menu command and upcoming shell dialogue. The colors can be saved and appear in the ⟨*input file*⟩ as eight digit hexadecimal integers. But it results in warnings for each of the detected color strings when restarting the program with this ⟨*input file*⟩ since the program default color scales will not be displayed any more.
Colors should not be changed since it took a long developing time to receive the different shadings of the scales that fit for almost all applications including a faded scale for recording video tapes (see 3.3 below). A general warning at this place: Don't waste too much time with generating your own colors.
The use of changing a specific color out of the scale is to point out a specific range by giving it an extreme color. It also is useful to further fade out the colors when recording video tapes, since colors often come out differently on the various TV screens.

3.3 Editable items

There is no access with menu commands to items that need not to be changed once they are adjusted for a certain problem. These items can only be changed by direct editing the ⟨*input file*⟩. They cannot be changed during execution of the program.

Scaling the domain

`FACTOR =` [REAL]

This item is the most important one to display the data in the correctly scaled manner such that they are visible with the standard view. This value is taken for an overall scaling of the domain. If the data for x- and y-coordinates are orders of magnitude away from unit length, you redefine the unit length with this value to the order of your data.
Note that this value does not affect the z-data. For flexibility in scaling z-data, the maximum value for each z-data set has to be provided in the data files (see 4.2.2 below). Scaling is done according to the maximum value detected for the specified z-data set over all files (time steps) of a sequence. This maximum is set to unit length by the program.
Here are examples of the effects of `FACTOR`:
In the default case (the x- and y-coordinates range between 0 and 1), this scaling factor is simply 1 (`FACTOR = 1.0`). If the x- and y-coordinates range between $0.02 < x < 0.05$ and $-0.04 < y < 0.02$ a `FACTOR = 0.01` results in a displayed range of 2 to 5 units for x and -4 to 2 units for y. A `FACTOR = 0.1` results in a displayed range of 0.2 to 0.5 units for x and -0.4 to 0.2 units for y.
If x- and y-coordinates differ by orders of magnitude, scale the domain by using another grid type value (yet only available for Cartesian grid). Thereby `FACTOR` will only affect the remaining axis not excluded by the grid type value (e.g. `GRID = 4, 6`), or will have no effect at all, scaling both x- and y-coordinates independently to their maximum value detected while reading the data file (for `GRID = 8`).

Handling of adjacent coordinate systems

To show a second set of z-data you might as well start the program a second time and place the two graphics windows aside to each other. But the advantage of displaying two coordinate systems simultaneously within the same graphics program reveals when an animation of a time series is displayed. Now you are able to compare the temporal development of two species (or values of z-data) synchronized with each other.

`NUMOAX = `[INTEGER]
1 one coordinate system is displayed (default),
2 two adjacent coordinate systems are displayed.
The shell based dialogues automatically ask for z-data sets for both coordinate systems, when this value is set to 2.

`SIDE = `[INTEGER]
This item will only be considered if `NUMOAX = 2`.
0 sets the second coordinate system behind the first,
1 juxtaposes the two coordinate systems (default).

`FSHIFT = `[REAL]

Set the spacing between the two coordinate systems, the default is 0.4 times unit length. This item will only be considered if `NUMOAX = 2`.

Length of the axes

`XLMIN =` [REAL]
`XLMAX =` [REAL]
`YLMIN =` [REAL]
`YLMAX =` [REAL]
`ZLMIN =` [REAL]
`ZLMAX =` [REAL]
Define the start and end point of the axis according to the predefined unit length. Every tenth of unit will be indicated by a little circle around the axis. The default axes are drawn from 0.0 to 1.1 times unit length.

Naming the axes

`XAXIS =` [CHAR STRING]
`YAXIS =` [CHAR STRING]
Specify the text to appear at the x- and y-axis, default strings are 'x' and 'y'.

Choosing initial viewing matrix

`STDOSPC =` [INTEGER]
This item corresponds with the viewing matrices described in 3.2 above.

1	starts up by using the standard view angle (default),
2	uses the view which is defined in `SPECIAL1`,
3	uses the view which is defined in `SPECIAL2`,
else	starts up by using the `LAST` viewing matrix.

Choosing a color scale

`COLOR =` [INTEGER]
The program provides four possible color scales to map a z-data set as color information to the displayed surface(s). It is possible to change the colors of the scale and save these changes to the ⟨*input file*⟩ (see 3.2 above). You might want to do so to pick out a special range by giving it an extreme color. Or you might want to fade out the red color since even the provided video scale is still too strong when recorded on a video tape. But if you saved the colors to the ⟨*input file*⟩, this results in warnings for each of the detected color strings when restarting the program with this ⟨*input file*⟩ since the program default color scales are not displayed any more.

-1 uses a damped temperature like scale for recording videos,
0 uses a temperature like scale from green to yellow (default),
1 uses a spectral like scale from infra red to ultra marine,
2 uses the inverse spectral scale.

Warnings

`PENALTY =` [INTEGER]
Set the time in seconds for which the program pauses while displaying warnings in the explanation window. The default penalty time is two seconds.

`SUPPRESS =` [INTEGER]
0 will not suppress warnings (default),
1 will suppress all warnings.

User supplied text

All the following text strings are displayed in the explanation window at the bottom of the screen.

`TEXT1 =` [CHAR STRING]
`TEXT2 =` [CHAR STRING]
`TEXT3 =` [CHAR STRING]
You may list your institution, name(s) and the date in here. Each string should not exceed 80 characters in total. It appears to the left of the color bar. To avoid collision, do not exceed a maximum of 40 characters.

`TEXT4 =` [CHAR STRING]
`TEXT5 =` [CHAR STRING]
These text strings are written below the left and right end of the color scale. Provide a minimum and maximum value if you want to notify these numerical values for representation purposes. Simply 'min' and 'max' (default) are more flexible and easy to read. If orbit data is concerned (for `GRID =` -1), type 't' for both text strings, resulting in t_0 and t_{end} to indicate time development.

`TEXT6 =` [CHAR STRING]
Text to be written to the right hand side of the color scale, e.g. 'Profile of:' or 'Phase portrait of:'.

3.4 Example of an input file

On the next page there is an example of an ⟨*input file*⟩ for showing two coordinate systems. The origin of the second one is placed 0.4 units away from the end point of the y-axis of the first coordinate system (at (0.0,1.4,0.0) in coordinates of the first system). The data is reflected at the y-axis. The graphical output is displayed in a window of the size required for recording on VCR tapes with PAL norm. No extra color information is given. The grid is displayed and the wire frame model will be antialiased with a linewidth of two pixels.

```
# begin of input
SUPPRESS  =  1        # suppress warnings
GRID      =  2        # 1=structured, 2=Cartesian
NUMOAX    =  2        # number of axis
SIDE      =  0        # 1=aside, 0 behind
FSHIFT    =  0.4E+00  # how far to shift
HALFX     =  2        # reflection at y-axis
HALFY     =  1        # no symmetry along x-axis
WIDTH     =  2        # no. of pixels for linewidth
ANTI      =  1        # 1=antialiasing, 0 no antialiasing
PENALTY   =  2        # time [s] for error messages
STDWIN    =  1        # 1=video size, 2=full screen
STDAXIS   =  1        # 1=standardaxis, -1=outlines
STDTEXT   =  1        # 1=standardtext by textbox.c
STDOSPC   =  0        # 1=standard view, 2=SPECIAL1
# viewing matrices
SPECIAL1  =           # Save views:  Special1
2.7066555023  -1.8550503254  -0.2804735005  -0.2804175019
1.5668778419   1.7294710875   0.6439903378   0.6438614726
0.5945240855   3.8873367310  -0.4203535914  -0.4202695191
0.0194530394  -0.0301657952   0.0422295257   0.0442208871
SPECIAL2  =           # Save views:  Special2
 <4×4 Matrix, see above>
LAST      =           # viewing matrix of last saving
 <4×4 Matrix, see above>
COLOR     =  0        # 0=temperacol, 1=spectralcol
HOW       =  -4       # display style; without grid
FACTOR    =  1.0E-02  # to scale the domain
# length of different axis
XLMIN     =  -0.5E+00
XLMAX     =  0.5E+00
YLMIN     =  0.0E+00
YLMAX     =  1.0E+00
ZLMIN     =  0.0E+00
ZLMAX     =  1.0E+00
# strings to write to the domain axis
XAXIS     =  r
YAXIS     =  z
```

continued on the next page

continued from the previous page

```
# list of different variables
CHOSE1    =  p      # CHOSE is for dialogue
NNNN      =  P(PA)  # NNNN=string identifier in data
ZAXIS     =  p[Pa]  # ZAXIS is for display
CHOSE2    =  R
NNNN      =  RHO
ZAXIS     =  rho
CHOSE3    =  u
...
...
# end of list of different variables
# text to display in the textbox
TEXT1     =  Interdisziplinaeres Zentrum
TEXT2     =  fuer Wissenschaftliches Rechnen
TEXT3     =  UNIVERSITAET HEIDELBERG 1994
TEXT4     =  min          # left end of the color bar
TEXT5     =  max          # right end of the color bar
TEXT6     =  Profile of:  # aside to the colorbar
# end of input
```

4 Data files

Animated sequences of time dependent solutions to systems of partial differential equations on a two dimensional grid was the central idea behind this program. Therefore every time step is written to a separate data file with a base name and an increasing number indicating temporal development of a vector valued solution. Nevertheless the program *cnom2.0* fits on a wide class of related problems.

The data can be provided in ASCII or binary format. Use the ASCII format as long as you need to look and read in your data files. If this is no longer necessary, skip to the binary format to save space on the disc and to speed up displaying the data.

Since *cnom2.0* can convert back and forth between both formats, the description of binary format (see 4.7 below) is only necessary, if your numerical output should directly be written in binary format in order to save space on the disc.

4.1 Data file names

The data file name is build up of a base name and a (two or more) digit number for the counter of the data files to be displayed in a (time) series one after another. When the grid changes or the time increases, store the data in a new file with the same base name but increased counter. In each file there is only one grid (x- and y-data) provided for this time step or file, but various function values (z-data sets) are possible. Animation of temporal development is done by loading the data files step by step. This feature can also be used for example to show a refinement or coarsening of grid for steady state solutions received with a multigrid method.

For a file number less than 10 a leading zero is required. This is done for convenience when listing the data files, such that data01 is followed by data02, instead of data1 being followed by data10, data100, data11, as is correct for lexicographical ordering. The numbers may exceed the two digits range, but the ordering of the listing then is destroyed. Make sure that you always provide all data files counting from data01, data02, data03, integer by integer. If the program detects a missing file before reaching the end number, the program exits to avoid a more serious error.

A file name does not have an extension when the file is in ASCII form, its binary form is marked by the extension '.bin'. If there is a data file with a '.bin' extension, this file will be preferred. The conversion of ASCII formatted data to binary data and vice versa can be done by menu commands, if there are no files with the destination names found in the same directory. For example the ASCII formatted files data01, data02, data03, can be converted to binary formatted files data01.bin, data02.bin, data03.bin. Removing of files must be done explicitly by yourself.

4.2 The general structure

Here is a description of the general structure of the data files.

Own comments on the data can be placed at the top of the data file, since the formatted reading of data does not start before the keyword `>DATA` (`>BINDATA` for a binary file) is found. Keywords are specific strings all beginning with a `>` sign. Values given behind the keywords are e.g. responsible for the size of allocated arrays or for the scaling of the data. The keyword `>FIN` closes this part of the data file. Now your x- y- and z-data, and if necessary, a list of elements for unstructured grid follow. A `<` sign marks the end of file (not necessary).

```
Own comments on data...
>DATA (>BINDATA)
...
>ICOUNT (>GRIDX >GRIDY >NV >NE)   (necessary keywords)
...
>MIN (>MAX >PARAMETER >TIME)      (optional keywords)
...
...
>FIN
>E (only for unstructured grid)
...
>X
...                                your data
>Y                                 (structure depends on
...                                the value for GRID
>Z                                 in the ⟨input file⟩)
...
...
<
```

Depending on what type of grid is announced in the ⟨*input file*⟩, the keywords for allocation of data arrays vary: `>GRIDX` and `>GRIDY` are expected for a grid type value between 0 and 10, `>NV` and `>NE` are expected for unstructured grid with a type value greater than 10 (which also requires a `>E` marked line followed by the internal numbers of vertices forming the elements, see 4.6 below). `>ICOUNT` is expected for orbit data (grid type value less than 0). `>TIME` and `>PARAMETER` statements are optional keywords.

4.2.1 Parameter statements

To display time and parameters of each data file in the explanation window they have to be supplied somewhere in the beginning of the data file. You may do so before the `>FIN` statement, such that it remains visible if the file is in binary form. At maximum two parameters are displayed (the first two behind the keyword) but you are free to write a whole list of parameters so that you only need to change the position of the keyword `>PARAMETER` in the data file to display another pair. The [CHAR STRING] at the beginning of the line is also printed in the explanation window, such that you can identify the selected parameters.
These statements are optional.

```
>TIME [REAL]
>PARAMETER
[CHAR STRING][REAL]
[CHAR STRING][REAL]
...
```

4.2.2 Scaling z-data

The reading of extreme values is separated from the reading of the rest of the data. This is because the program determines the absolute extreme value over the whole time series out of the extreme values related to each of the files of this series. A time series is specified during a dialogue at the program start or restart when giving the end counter of data files. At the program start, only the specified z-data sets of the first file are read and automatically scaled in a way that their absolute extreme value (or their absolute amplitude) over all time steps appears as unit. While animating the whole series, the z-data reach their extreme value in at least one data file and at least at one point of the grid. When reading a specific file by specifying its number in a dialogue, or with a menu command, no reading of extreme values is done, since the absolute extreme values are already known for this time series. Scaling is done automatically.

Here is a short example of what does it mean: Consider a small perturbation of a zero solution as initial data that grows slowly in the beginning but then tends to explosion. The z-data sets will be scaled to a much smaller absolute extremum when considered only for a short time series, as would be the case if the time series is very long. But you might wish to have a closer look to what happens in the beginning, so you would only load a short series. Possibly also this doesn't reveal what really happens, since the perturbations are all positive and of almost the same size. Now scaling to the maximal amplitude will force the minimum value to be zero and the maximum value to be one (the unit).

This second scaling method is indicated during the dialogue at the program start or restart with a dash in front of the abbreviation used to select a z-data set.

Supply the extreme values in each data file between the keywords `>MAX`, `>MIN` and `>FIN`, at the beginning of the file. To scale the desired z-data set to its maximum or to its maximal amplitude over all loaded data files, it is necessary to supply the extreme values for all possible z-data sets in each data file. During the preparational reading the program searches for the keywords `>MAX`, `>MIN`

and `>FIN`, expecting to find the identifier for the required z-data set followed by a [REAL] extreme value on the next line. It is not essential which of the keywords `>MIN` or `>MAX` (each followed by the identifiers and values) appears first, but if the specified identifier followed by an appropriate value is not found before detecting the keyword `>FIN`, the program exits with an ERROR message.
Here is the part of the data file handling extreme values. Exactly this part is read by the preparational reading routine.

```
>MIN
⟨Identifier of z-data set 1 ⟩
[REAL]
⟨Identifier of z-data set 2 ⟩
[REAL]
...
...
⟨Identifier of z-data set n⟩
[REAL]
>MAX
⟨Identifier of z-data set 1 ⟩
[REAL]
⟨Identifier of z-data set 2 ⟩
[REAL]
...
...
⟨Identifier of z-data set n⟩
[REAL]
```

4.3 Orbits

To display orbital data it is not necessary to define a net or grid (set `GRID = -1` in the ⟨*input file*⟩). The program only searches for the keyword `>ICOUNT`, reads the integer behind this keyword in order to allocate the data array. This block closes again with a line `>FIN`. Then the program expects to find the (x, y, z)-values in the data file behind a line marked by `>P2` (for 2D-data), or `>P3` (for 3D-data), and interprets them as curves.
Just one curve can be stored in one file and the number of files counts the different curves. Each curve then is colored from the first to the last point (e.g. indicating temporal development of an orbit).

In contrast to the other problems the orbit representation hasn't yet an automatic scaling and no animation is done to show temporal development. All data files as requested at the program start or restart will be displayed at once.

```
ODE solution for a dynamical system in the plane with four
parameters
stiffness for small eps
>DATA
>ICOUNT 100
>PARAMETER
l:  5.0E+02
s:  4.0E+00
k:  3.0E+00
eps:  5.0E-04
>FIN
>P2
[2 × REAL]
[2 × REAL]
[2 × REAL]
...
...
<
```

The data files look like in the box above. For three dimensional data simply change `>P2` into `>P3` and supply [3 × REAL] on each line.

4.4 Cartesian grid

The Cartesian grid means perpendicular x- and y-coordinates somewhere in the plane and not necessarily equidistant. Since x- and y-coordinates are perpendicular, it is only necessary to define x- and y-partitions. The grid is constructed by the cross product of those. For allocation of the arrays supply the number of these partitions behind a keyword `>GRIDX` or `>GRIDY`, respectively. In the following these numbers are shortly called `GRIDX` and `GRIDY`. A `>MIN/MAX` statement has to be supplied (see 4.2.2 above), `>PARAMETER` and `>TIME` are optional, before the keyword `>FIN` closes this block.
A line marked by `>X` or `>Y` has to be followed by the x-coordinates or the y-coordinates, respectively. The z-coordinates for each set of z-data must be given behind a `>Z` marked line in slices of `GRIDX` entries for each y-coordinate. Each slice is introduced by its identifier (the `NNNN` entry specified in the ⟨*input file*⟩). This looks as follows:

>Z

$$\left.\begin{array}{l} \langle \textit{Identifier of z-data set 1}\rangle \\ [\texttt{GRIDX} \times \text{REAL}] \\ \langle \textit{Identifier of z-data set 2}\rangle \\ [\texttt{GRIDX} \times \text{REAL}] \\ \cdots \\ \langle \textit{Identifier of z-data set n}\rangle \\ [\texttt{GRIDX} \times \text{REAL}] \end{array}\right\} \times \texttt{GRIDY}$$

...
...
<

The structure consisting of identifier and slice of data parallel to the x-axis has to be written for each y-coordinate, which is `GRIDY` times in total. It is absolutely necessary to find the identifier for each slice of x-data. This can also be done as follows:

>Z

$$\left.\begin{array}{l} \langle \textit{Identifier of z-data set 1}\rangle \\ [\texttt{GRIDX} \times \text{REAL}] \\ \cdots \\ \langle \textit{Identifier of z-data set 1}\rangle \\ [\texttt{GRIDX} \times \text{REAL}] \end{array}\right\} (\texttt{GRIDY}\text{ in total})$$

...
...

$$\left.\begin{array}{l} \langle \textit{Identifier of z-data set n}\rangle \\ [\texttt{GRIDX} \times \text{REAL}] \\ \cdots \\ \langle \textit{Identifier of z-data set n}\rangle \\ [\texttt{GRIDX} \times \text{REAL}] \end{array}\right\} (\texttt{GRIDY}\text{ in total})$$

<

The ordering within one set of z-data should correspond to the ordering of y-coordinates.

Scaling the domain

There are different possibilities of scaling the domain. The according flags have to be provided in the ⟨*input file*⟩, where the value behind the item `FACTOR` will

do an overall scaling. It should be used to scale to the appropriate order of magnitude. But this might not be sufficient, since x- and y-data may differ by orders of magnitude, so that you want to scale the axis independently. Therefore change the grid type value in the ⟨*input file*⟩.
All positive even integers less than ten are accepted for Cartesian grid. Both x- and y-data are divided by the value of `FACTOR` for `GRID = 2`.
When `GRID = 4` (`GRID = 6`) only y-data (x-data) are divided by this value, whereas the x-data (y-data) are scaled with the maximum of the absolute values of the first and the last x-coordinate (y-coordinate) given in each data file. So the unit length to the positive or negative side of the x-axis (y-axis) appears as maximal coordinate of the grid.
For `GRID = 8` both x- and y-data are scaled with the maximum of the absolute values of the first and the last x- and y-coordinate.
See also the description in 3.1 for the items of the ⟨*input file*⟩ above.

Example of a data file

Here is an example of a data file. You may also look into the data files provided in the examples directory.

```
Neumann Randwerte, Fortsetzung aus FFNAME = start/symm50
>DATA
>TIME(S) 0.5000000E+00
>GRIDX 7
>GRIDY 10
>PARAMETER
d1:  5.0000000E-04
d2:  0.0000000E+00
>MIN
CO
0.0000000E+00
ACT
0.0000000E+00
>MAX
CO
0.1000000E+01
ACT
0.1000000E+01
>FIN
>X
0.0000000E+00 0.2040816E-01 0.4081633E-01 0.6122449E-01
0.8163265E-01 0.1020408E+00 0.1224490E+00
>Y
0.0000000E+00 0.2040816E-01 0.4081633E-01 0.6122449E-01
0.8163265E-01 0.1020408E+00 0.1224490E+00 0.1428571E+00
0.1836735E+00 0.2040816E+00
>Z
CO
0.3000000E+00 0.3010268E+00 0.3025640E+00 0.3051191E+00
0.3501093E+00 0.3409501E+00 0.3600909E+00
ACT
0.3005137E+00 0.3020525E+00 0.3046102E+00 0.3081763E+00
0.3596385E+00 0.3496466E+00 0.3704127E+00
CO
0.2956000E+00 0.2450268E+00 0.2355640E+00 0.2351191E+00
0.3501093E+00 0.3409501E+00 0.3600909E+00
ACT
0.3004447E+00 0.3034225E+00 0.3006102E+00 0.3000763E+00
0.3596385E+00 0.3496466E+00 0.3704127E+00
CO
0.2956000E+00 0.2450268E+00 0.2355640E+00 0.2351191E+00
...
<
```

4.5 Structured grid

Structured grid means all grids topologically equivalent to Cartesian grids. The grid type value in the ⟨*input file*⟩ is 1 (`GRID = 1`). It differs from the description above by the necessity of explicitly stating the x- and y-positions of all grid points, because the coordinates of the grid points cannot be determined by the cross product any more.

The formatted reading again starts with a line `>DATA`, for allocation of arrays supply the number of partitions after `>GRIDX` and `>GRIDY`. A `>MIN/MAX` statement has to be supplied (see 4.2.2 above), `>PARAMETER` and `>TIME` are optional, and this block closes with `>FIN`.

The `>X` and `>Y` marked lines are followed by all x- and y-data of the grid points, respectively. Since the grid is still structured, the ordering is the same as for Cartesian grid, that means `GRIDX` × x-coordinates for the first y-row are followed by `GRIDX` × x-coordinates for the second y-row, the third, and so on until `GRIDY` rows are complete. The y-coordinates must be supplied in exactly the same way.

Now the sets of z-data have to be provided in the same way as for Cartesian grid (see 4.4 above). Note that for the Cartesian and structured grid it is absolutely necessary to find the z-data set identifier (corresponding to the `NNNN` entry in the ⟨*input file*⟩) for each slice of x-data.

Scaling of the x-data independently of the y-data (as was explained for Cartesian grid) is not possible. Only overall scaling can be done with the value of `FACTOR`, anything else must be done at runtime (zooming in or out on a single axis).

4.6 Unstructured grid

Unstructured grid means all two dimensional grids consisting of triangles (the grid type value in the ⟨*input file*⟩ is `GRID = 11`) or mixed triangles and rectangles (`GRID = 12`). These elements (triangles and rectangles) need not be connected, although a vertex (or node) might be shared by numerous elements.

For the data format this means, that the number of vertices must be specified behind the keyword `>NV`, and the number of elements must be given behind the keyword `>NE`. In the part after `>FIN` provide the list describing the connections of the vertices to form all elements. This list must be given behind a keyword `>E`. Every single element has to be defined by integer numbers of the vertices of this element on a single line. For `GRID = 12`, the carriage return is taken to specify whether it is a triangular or rectangular element.

```
>E
a1 b1 c1 (d1)
a2 b2 c2
...
```

Those integers a_n b_n c_n (d_n) refer to the internal numbering of the data, e.g. the n^{th} element is build out of three (four) vertices constructed by the ${a_n}^{\text{th}}$ x-, y- and z-coordinate as first vertex, the ${b_n}^{\text{th}}$ x-, y- and z-coordinate as second, and so on.

```
>DATA
>TIME [REAL]
>NV [INTEGER]
>NE [INTEGER]
>MIN
...
...
>MAX
...
...
>FIN
>E
[NE × 3 (4) × INTEGER]
>X
[NV × REAL]
>Y
[NV × REAL]
>Z
⟨Identifier of z-data set 1 ⟩
[NV × REAL]
⟨Identifier of z-data set 2 ⟩
[NV × REAL]
...
...
⟨Identifier of z-data set n ⟩
[NV × REAL]
<
```

Now the data file should look like above. You may place your own comments on the data in top of the file before the keyword `>DATA`. The `>TIME` and

`>PARAMETER` statements are optional. Then supply a line with `>NV`, a line with `>NE`, and the `>MIN` and `>MAX` statements. Up to a line `>FIN` the preparational reading takes place.
Now provide the list of elements behind a `>E` marked line. The coordinates of those nodes must be listed behind the corresponding marks, that means a `>X` for the x-coordinates, a `>Y` marked line for the y-coordinates, and a `>Z` marked line for the z-coordinates.
There can be more than one set of z-data to be represented in one data file, using different identifiers (as already explained in 4.2.2). This format differs from those above since the x-, y-, z-coordinates of the elements appear all at once after the `>X`, `>Y` and `>Z` lines, respectively, so there is only a single identifier per z-data set n and data file.

4.7 From ASCII to binary data

As long as you want to edit the data it is appropriate to handle ASCII files. To save space on the disk and to fasten up the animation, it is more convenient to provide the data in binary form. *cnom2.0* is able to convert between formatted ASCII and binary files back and forth just by menu driven command for all of the grid types described in 4.3 until 4.6. The file names do not have an extension, when they are of ASCII type and can be converted into binary form with the same name together with a '.bin' extension. To convert into ASCII format, there has to be an appropriate file with '.bin' extension. No conversion is done, if there is already a file with the destination name in the same directory. The program does not remove any data files − this must be done explicitly by yourself.
During conversion into binary form comments are only saved for the first file of the specified sequence. The part up to `>FIN` will rest in ASCII form for all data files to enable the user to look for the right data and to detect errors resulting from a wrongly adjusted ⟨*input file*⟩.
If the output shall be given directly as binary data for reasons of space for example, here is a description how it should look like.
The formatted reading of binary files start behind the keyword `>BINDATA` and remains in ASCII up to the string `>FIN`.
Behind this line it follows an ASCII printed number of different sets of z-data, then ASCII printed identifiers of all sets, each on a single line, binary packed x-coordinates, binary packed y-coordinates, and for unstructured grid also binary packed elements. Finally the binary packed z-coordinates are stored, for each set of z-data in the order defined by the order of the identifiers introduced above.

...
(the same as in ASCII form)
...
>FIN
[INTEGER]
[n × CHAR STRING]
[BINARY DOUBLES]
[BINARY DOUBLES]
[BINARY INTEGERS]
[n × BINARY STRUCT]

The number of binary doubles or integers corresponds to the grid type (see 3.1 for `GRID`). Elements have to be provided only for grid types greater than ten. The binary packed elements appear transposed to the description in 4.6, that means they are written as [`NE` × BINARY INTEGER] the first vertex numbers, [`NE` × BINARY INTEGER] the second vertex numbers, then [`NE` × BINARY INTEGER] the third vertex numbers, and only for `GRID = 12` [`NE` × BINARY INTEGER] the fourth vertex numbers, indicating a triangular element by -1.
Each binary pack of z-data sets is structured as follows:
At first two floats for minimum and maximum value of the data are stored. After that all data is stored as binary integers (16 bit) by approximating them between minimum and maximum, using following formula (this results in a medium error of 10^{-5}, which is too small for a computer display to worry about):

$$\text{new data} = 65535 * \frac{\text{data} - \text{min}}{\text{diff}}$$

where $\text{diff} = \text{max} - \text{min}$ (diff is set to 1, if max equals min).
Orbit data is just put to binary form as doubles without additional scaling.

5 Using mouse and keyboard

The program is mainly menu driven using the mouse. The menu pops up on a RIGHTmouse button button click while the cursor is currently within one of the graphics windows. Clicking again somewhere in the window without having highlighted a command makes the menu disappear without changing the displayed objects. Highlighting a menu line with a triangle invokes a nested submenu. A menu line without triangle is a direct command. For frequently used commands there are short key defines. They are shown in parenthesized letters behind the corresponding menu entry, e.g. (u) for Undo. For some

commands the program will need a shell window for questioning the user. The graphics window is pushed to the back when expecting shell input and pops automatically to the front when the input is accepted. On IBM machines make sure to activate the window by clicking on the border.
Some commands use the explanation window to display error messages, to ask for number keys as input, or to supply a blue bar as scale for transformation routines. After passing the penalty time or getting the right input, this window displays again the user defined text and color bar.

5.1 The commands

Here a brief description of the pop-up menu commands is given. The commands in this description are printed in *slanted style*. The words in capital letters (not slanted) refer to the items of the ⟨*input file*⟩ like for example `HOW` (see 3).

EXIT ▷ Normal way of exiting the program. The settings achieved interactively can be saved in the ⟨*input file*⟩.

- *Save all* Save all settings in their current state.
- *Save views* Just save the current viewing matrices (into the slots `SPECIAL1`, `SPECIAL2` and `LAST` of the ⟨*input file*⟩).
- *Save colors* Just save the current colors of background, text and scale.
- *Escape (ESC)* Do not save anything at all.

Store views ▷ Store user defined viewing matrices in

- *Special 1*
- *Special 2* or
- *From* (interacts with *Video* ▷ *Interpol-From*).

Storing just means to keep the matrices during this run of *cnom2.0*. They can be retrieved by the *Restore views* ▷ submenu. The viewing matrices `SPECIAL1(2)` will be saved to the ⟨*input file*⟩ when exiting through *EXIT* ▷ *Save all* and *EXIT* ▷ *Save views*. Otherwise they are lost.

Curves ▷ Specify how to interpolate between nodes (see also `HOW`).

- *Splines* ▷ Bring up a submenu for different types of supported spline interpolation.

  Bezier type

- *Cardinal type*
- *B-type* and
- *NURBS* Non Uniform Rational B-splines.

Polygon (i) Linear interpolation using Gouraud shading.

Patches (o) Gouraud shading of segments between the nodes.

Black & white $\triangleright$ Black and white output might be desired for printing.

- *Background* Toggle background and text color.
- *B-spline* B-spline in text color.
- *Polygon* Polygon in text color.

Display $\triangleright$ These commands adjust the surroundings.

- *Symmetry* $\triangleright$
  - $\langle x \rangle$*-axis* Toggle reflection of the data at the x-axis (see `HALFX`).
  - $\langle y \rangle$*-axis* Toggle reflection of the data at the y-axis (see `HALFY`).
- *Periodicity* $\triangleright$
  - $\langle x \rangle$*-axis* Toggle periodicity at the x-axis (see `HALFX`).
  - $\langle y \rangle$*-axis* Toggle periodicity at the y-axis (see `HALFY`).
- *Axes (x)* Toggle visibility of the axes (see `STDAXIS`).
- *Grid (g)* Toggle visibility of the grid (see `HOW`).
- *Linewidth (w)* Set thickness of lines in pixles by number keys (see `WIDTH`).
- *1/4 or full* Toggle between a smaller window to be used when recording an animation onto VCR tape (768×576 pixels (output area for PAL norm) or 646×486 pixels (output area for NTSC norm)) and full screen display (see `STDWIN`).
- *Antialiasing* Toggle the smoothness of lines to avoid jaggies tuning the line color with the background color (see `ANTI`).
- *Large font* Toggle the text modus (see `STDTEXT`).

Video $\triangleright$ These commands can be used to create animation scripts (see 6.1.1 below).

Videotext	Bring up the text and color scale of the explanation window to the window with the graphics output for at least three seconds. This time can be extended by the sleeping time parameter provided within the *Dialogue* ▷ *Video values*.
Sleep	Pause for a time period which is to be set within *Dialogue* ▷ *Video values*.
Roll	Animates a rotation using values to be defined within *Dialogue* ▷ *Video values*.
Interpol-From	Animate a linear interpolation between a previously stored viewing matrix to be set within the *Store views* ▷ *From* command and the current viewing matrix.
Ahead (a)	Play a forward sequence through all files from the current to the last specified data file.
Backward (b)	Play backwards to the first data file.

Script editor ▷ These commands will result in pushing the graphics window to the back and waiting for keyboard input to specify the scenes to be edited.

Smooth	Smooth between adjacent scenes of the current ⟨*script*⟩.
Delete	Delete scenes of the current ⟨*script*⟩.
Write ⟨*script*⟩	Record the last scene into the current ⟨*script*⟩.
Open ⟨*new*⟩	Here you have the choice of loading another file. The file will be created if non-existing.

Dialogue ▷ These commands will result in pushing the graphics window to the back and waiting for keyboard input.

Read ⟨*script*⟩	Invoke a dialogue to ask for the first and last scene number to be read out of the ⟨*script*⟩. The default name is 'script', but there can be specified another file, using the -s command line option at the program start or using the command *Script editor* ▷ *Open* ⟨*new*⟩.
Video values	Change the values considered for animation in the scenes. For detailed explanation see 6.1.2 below.
New species	Get other spieces into the current view.
Restart	Reactivate the startup dialogue.

Data to binary	Convert a sequence of files into binary form.
Data to ASCII	Convert a sequence of files into ASCII form.

Translate ▷ Translates the object in direction of one of the axis. These commands and the following two submenus invoke a blue bar in the explanation window. That is a ruler for a brief orientation. To perform a transformation, click on the RIGHTmouse button and hold it. The x-screen coordinate of the cursor anywhere on the screen will be used to transform the object according to the values in the bar, while the RIGHTmouse button is pressed. Releasing the RIGHTmouse button defines this view to be the current viewing matrix and returns to the level of the pop-up menu.

⟨*x*⟩*-axis*	Translate along the x-axis.
⟨*y*⟩*-axis*	Translate along the y-axis.
⟨*z*⟩*-axis*	Translate along the z-axis.

Rotate ▷ Rotates the object around one of the axis.

⟨*x*⟩*-axis*	Rotate around the x-axis.
⟨*y*⟩*-axis*	Rotate around the y-axis.
⟨*z*⟩*-axis*	Rotate around the z-axis.

Zoom ▷ Zooms in or out on a single axis or on all axes.

⟨*x*⟩*-axis*	Scale the x-axis.
⟨*y*⟩*-axis*	Scale the y-axis.
⟨*z*⟩*-axis*	Scale the y-axis.
all axes	Scale all axes.

Restore views ▷ Retrieve the previously stored viewing matrices.

Initial state	Set the state of startup configuration, as specified in the ⟨*input file*⟩ (see `STDOSPC`).
Standard	Set the standard viewing matrix.
Special1 *Special2* *From*	Set views previously stored during this run or defined in the ⟨*input file*⟩.

Data steps ▷

First (f)	Load first file of the sequence.
Previous (p)	Load previous file of the sequence.
Next (n)	Load next file of the sequence.
Last (l)	Load last file of the sequence.

Select (s) Load a file selected by number keys. Confirm the input by pressing ⟨ENTER⟩. Just pressing ⟨ENTER⟩ might terminate the program if there is no data file named ⟨*base name*⟩00. Typing a wrong number can be corrected with ⟨BACKSPACE⟩. Looking for a file with a counter greater than the maximal counter given at the program start or restart yields in a warning message.

Colors ▷ Changing of colors can be done by indicating integer values between 0 and 255 for red, green and blue according to RGBmode. Pressing ⟨ENTER⟩ leaves the colors unchanged.

Maincolors Background color and text color can be changed.

Scale All colors of the scale can be changed by first specifying the number of their appearance counted from the left end of the bar.

Undo (u) Retrieve the state before the last command (toggling when repeated).

Repeat (r) Repeat the last command.

5.2 Direct transformation

It is also possible to use the LEFTmouse button and MIDDLEmouse button for a direct transformation without calling the pop-up menu and the blue bar. Hereby the x- and y-screen coordinate relatively to the position of the cursor when first pressing the button is used to perform the transformation. The LEFTmouse button will translate the object according to the x- and z-axis of the non-rotated standard view, the MIDDLEmouse button zooms in and out. Pressing both buttons will rotate the object. This steering requires some practice as well as the implementation needs to be improved.

6 Animation

A main purpose of the program is to achieve an easy way to run an animation. Therefore a script facility has been developed, that can be used either for on-line recording on VCR tapes or for presentation purposes. This script facility allows the user to write an animation script interactively. That means that single scenes can be invoked by menu commands and then be saved to a specified position in a ⟨*script*⟩, adding up to a complex movie of transformed and animated sequences of various data files.

6.1 Writing animation scripts

For writing animation scripts interactively, the submenus of *Video* ▷, *Script editor* ▷ and *Dialogue* ▷ can be used. You can also edit these script files, but be careful (see 6.2 below).
At first adjust the display in the desired way. That includes the display style with or without grid, axes, antialiased mode and so on. It is not important which window size is taken, so there is no problem of using the same script for different sizes of graphics windows. You can show the same animation on a full screen or just on NTSC or PALnorm size for on-line recording of VCR tapes.
Now run a command of the *Video* ▷ submenu, like for instance *Roll*. Write this last scene to the specified file by invoking *Script editor* ▷ *Write* ⟨*script*⟩. A dialogue leads through the next steps asking for the number of the scene to be edited. If it is greater than the number of already existing scenes, the new scene is appended, otherwise you may specify whether you want to insert (i), change (c) or skip writing ⟨ENTER⟩ this scene. Skipping leaves the ⟨*script*⟩ unchanged.
If a command needs additional input from mouse or keyboard, it does not make sense to save it as a scene to the ⟨*script*⟩. For these commands, you get a warning message in the shell. Interactively editable scenes are at least those under the *Video* ▷ submenu.
If you succeeded in writing a scene to the ⟨*script*⟩, retrieve this scene by the command *Dialogue* ▷ *Read* ⟨*script*⟩ and enter the same number you gave to the scene.

6.1.1 Interactively editable commands

At the beginning of a sequence of scenes it is often desired to record as well the color scale and the information provided by the explanation window. The command *Video* ▷ *Videotext* brings this information up to the graphics window, where it will rest for at least three seconds. This time is extended by the value considered within the *Sleep* command. *Video* ▷ *Sleep* is a command to fix a scene for a specified time. This time can be changed with *Dialogue* ▷ *Video values*. The *Sleep* scene might come out as one of the most often used scenes, since every newly adjusted display should rest a while so that the audience can follow.
To animate translations or zooming, store the current viewing matrix in *Store*

views ▷ *From*, perform the desired transformation and then invoke *Video* ▷ *Interpol-From*. This yields a linear interpolation between the viewing matrix previously stored in *Store views* ▷ *From* to the actual viewing matrix. Notice that linear interpolation does not work for rotation. So any rotation performed after storing the viewing matrix *From* might give funny results for interpolation. Use the *Video* ▷ *Roll* command instead to perform rotations correctly. It corresponds with *Dialogue* ▷ *Video values* x-, y-, z-angle values measured in degree per frame. So multiply the number of frames with these angle values to get the total angle of rotation for this scene. The default setting is a counter-clockwise quarter rotation around the z-axis.
The commands *Video* ▷ *Ahead* and *Video* ▷ *Backward* run a forward or backward sequence of data files from the current to the first or the last of the specified files given by the counter in the command line at the program start with the option -c. If fewer files shall be animated, the number can be specified with *Dialogue* ▷ *Video values*.
Script editor ▷ *Smooth* interpolates linearly between two adjacent scenes of the ⟨*script*⟩, which is useful after deleting some scenes or after abrupt jumps for other reasons.
Deleting scenes can be done with *Script editor* ▷ *Delete* asking for the first and including last scene number to be deleted.

6.1.2 Interaction with the *Dialogue* ▷ menu

In order to alter the constants used for the *Video* commands, use *Dialogue* ▷ *Video values*. It calls up a shell dialogue like follows:

```
Enter sleep time in seconds (actual [INTEGER]):

Enter number of data files to be animated (ahead or backward) (actual [INTEGER]):

Enter number of frames for a scene (actual [INTEGER]):

Enter x-, y-, z-angle in degree/frame (actual [REAL] [REAL] [REAL]):
```

Pressing ⟨ENTER⟩ does not change the values and passes to the next line.
The sleep time is used for the *Video* ▷ *Videotext* and *Sleep* commands.
The number of data files affects the internal counter for an animation of successive data files. If you do not want to run up to the last (first) of the loaded

data files all at once (which is the default),you have to change this number. It is used for the *Ahead (a)* and *Backward (b)* commands.
The number of frames is considered for interpolation of viewing matrices (*Interpol-From*, *Smooth*) and for the *Roll* command. For the interpolation the begin and end point of the transformation is known and the number of frames gives the number of intermediate steps. For the rolling, in each frame a rotation with the specified x-, y-, and z-angle is performed, so the default setting of 30 frames with (0.0,0.0,3.0) as angle parameter yield a quarter (90°) counter-clockwise rotation around the z-axis.
You can load other sets of z-data by *Dialogue* ▹ *New species*. With *Dialogue* ▹ *Restart*, you can load another ⟨*input file*⟩ and other base name of data files, not necessarily in the same directory, if their path is specified. Then continue the script writing.
The animation of ahead and backward played files can be fastened a lot if binary files are used instead of ASCII files (binary files are preferred if there are any with the specified base name in the current directory).

6.2 Editing a script

It is also possible to edit the ⟨*script*⟩ and copy or change scenes within the editor. Comments again begin with a # sign. The scenes are separated by dashed lines and a number is appended to each string 'scene'. The numbering once more is only for convenience and has no effect on the program, which always counts the 'scene' entries from the beginning of the file to determine which scene is desired. But writing a new scene interactively at the end will reset the numbering according to the appearance of the scenes.
The number behind the equal sign of the line beginning with the string 'scene' refers to a command of the graphics program.
It turned out to be very helpful to extend the ⟨*script*⟩ by a heading comment on the date, occasion, and the base names and counters of the necessary data files to run the script.
The order of the items in each scene is not essential, and even most of the values might not be used for the specific command that runs this scene. The items resemble those of the ⟨*input file*⟩ and can be looked up in 3. Additional strings are listed here, their values may be changed on own risk.

`scene`n	= [INTEGER]	Reference number for the n^{th} scene command.
`abit`	= [INTEGER]	Sleep time in seconds.
`retry`	= [INTEGER]	Number of file steps to run up to.
`count`	= [INTEGER]	Number of current file.
`countil`	= [INTEGER]	End number of files to determine overall extreme values.

`frame`	= [INTEGER]	Number of frames for a transformation scene.
`turn`	= [3 × REAL]	Turn angle for the *Video* ▷ *Roll* command.
`color`	= [CHAR STRING]	Dialogue expressions for z-data set to color the surface.
`ccc1`	= [CHAR STRING]	Dialogue expressions for z-data set 1.
`ccc2`	= [CHAR STRING]	Dialogue expressions for z-data set 2.
`prefix`	= [CHAR STRING]	Base name of data files.
`STATUS`	= [4×4×REAL]	Viewing matrix to define the begin of this scene.
`TO`	= [4×4×REAL]	A command corresponding to a transformation needs also a viewing matrix to define the end.

6.2.1 Example of a scene

This is for example how an interpolation scene looks like, when editing your ⟨*script*⟩:

```
#-------------------------------------------------------------
 scene3   =  77
 input    =  data/input
 color    =  c
 ccc1     =  h
 halfx    =  2
 halfy    =  2
 width    =  1
 anti     =  1
 stdaxis  =  8
 stdtext  =  1
 how      =  -5
 abit     =  1
 retry    =  0
 count    =  1
 countil  =  2
 prefix   =  oct
 frame    =  30
 turn     =  0.000000 0.000000 3.000000
 STATUS   =
 -4.4888792038  -0.0174074173  -0.0055919588  -0.0055918992
  0.0261135120  -2.9922599792  -0.9613670707  -0.9611728787
  0.0000000000   5.4500327110  -0.5276193619  -0.5275136232
 -4.0418205261  -5.2632784843  12.6291999817  13.2270812988
 TO       =
 -4.4894843102  -0.0174077749  -0.0055919588  -0.0055920780
  0.0261135120  -2.9926598072  -0.9614956379  -0.9613021612
  0.0000000000   5.4500327110  -0.5276193619  -0.5275136232
 12.6865949631  -2.0438196659  15.0700845718  15.6669950485
#-------------------------------------------------------------
```

6.3 Running a script

Invoke the *Dialogue* ▷ *Read* ⟨*script*⟩ command. The shell pops up to ask for the begin and end number of scenes to run. While running a script a little window indicating the scene number pops up. A once running script cannot be interrupted. After the last scene is displayed, there will be printed the word 'end' across the scene number. This also indicates the last available scene in case the end number exceeded the number of scenes found. If already the begin number is greater, then the graphics output is directed to the little window (a funny bug). Try again with a smaller number or another ⟨*script*⟩ invoked by *Script editor* ▹ *Open* ⟨*new*⟩.

7 Additional hints

Here are some desirable hints to experience a different view onto the data. For this you have to make some additional changes in the ⟨*input file*⟩ or directly in the data files.

7.1 Displaying time slices of one dimensional data

Numerical solutions of systems of differential equations that are one dimensional in space but developing in time can be displayed as a surface over a grid consisting of a space and a time axis. The grid type value should be chosen as indicated for Cartesian grid, if the position of each node on the space dimension does not change. Neither the space nor time step needs to be equidistant. Provide these positions behind the `>X` mark and the time for each solution behind the `>Y` mark. The file step opportunity now is no longer needed for animation of temporal development and so is free to show for example simulations on finer discretizations in space or time or just simulations varying other internal parameters.
The grid type value should be chosen as indicated for structured grid, if the position of the nodes on the space dimension do change. Now all of these positions have to be provided behind the `>X` mark, going through the slices from left to right. Behind the `>Y` mark every time step now has to be printed as often as the number of nodes indicates.
If as well the number of nodes changes with every time step, the grid type value should be chosen as indicated for unstructured grid, needing as well a list of elements to connect the grid points.

7.2 Two in one

To show two surfaces on top of each other, the two coordinate systems can be superposed. Set the value of `NUMOAX = 2`, and choose for `SIDE = 1`. `FSHIFT` then has to compensate for the maximal x-position, defined by `XLMAX`, so give it the value of `XLMAX` multiplied by -1. The same result is reached for `SIDE = 0` and `FSHIFT` as negative as `YLMAX`. You may not want to display the axes any more since both character strings for the z-values are written on top of each other. Therefore you can set `STDAXIS = 0`. Another possibility of avoiding collision of those strings but still displaying the axes can be done with a line
`ZAXIS =`

without entry for the displayed z-data sets. If you want to rectify the surfaces from each other, play along with wrongly adjusted maximum values in the data file to result in different scalings. Twice the maximum value for one set of z-data often will do. This manipulation of maximum values has to be done in only one of the data files of a sequence, since the maximum of all relative maxima is taken to scale the z-values (see 4.2.2 above).

7.3 Animating arbitrary objects

To use the video facility it might be useful to represent these objects build out of triangles or mixed triangles and rectangles in the described format for unstructured grid (see 4.6 above). You then can provide a color to it by arranging a z-value between 0.0 and 15.0, giving it a minimum value 0.0 and a maximum value 15.0, and you have access to the full color scale (choosing temperature or spectral color set) to color the given vertices. See for example in the directory ./examples/castel for the result displaying fractal solutions to the ground plan of the Castel del Monte.