GnuplotClass for FORTRAN Documentation

Lukasz A. Drozd *

August 2018

*Standard GNU license applies.

Disclaimer and copyright

This document contains documentation for the GnuplotClass.

1 The basics

1.1 What is gnuplotClass?

GnuplotClass generates high quality plots directly from FORTRAN code using command-line open-source gnuplot application. Figure 1 provides an example of a pdf generated using gnuplotClass.

Note that gnuplot must be installed prior to using gnuplotClass. It can be downloaded from `http://www.gnuplot.info/download.html`.

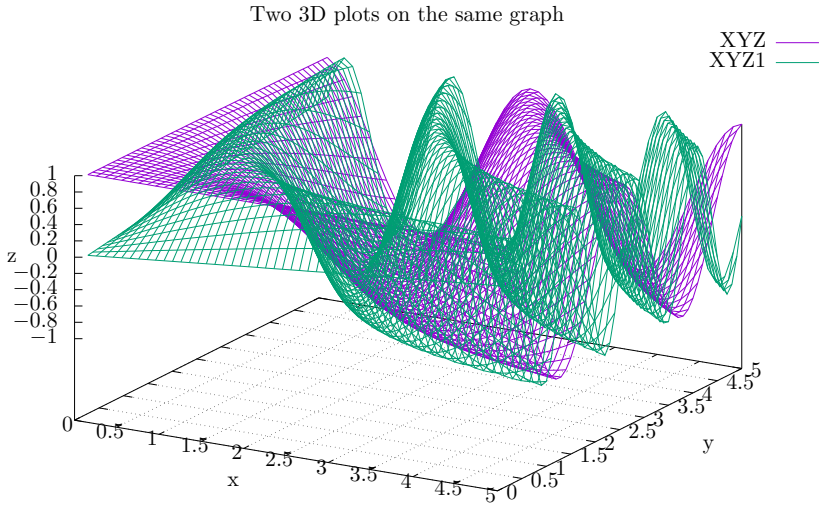

Figure 1: Sample plot generated by gnuplotClass and gnuplot.

Using the class does not require knowing gnuplot syntax but it is helpful. Gnuplot documentation can found at `http://www.gnuplot.info/docs_5.2/Gnuplot_5.2.pdf`. For a gentl(er) introduction, refer to `https://www.ethz.ch/content/dam/ethz/special-interest/gess/computational-social-science-dam/documents/education/Spring2017/Data_science/gnuplot.pdf` or P. K. Janert. "*Gnuplot in Action: Understanding Data with Graphs,*" Manning Publications Co., Greenwich, CT, USA, 2009, available free of charge at

http://www-bs2.informatik.uni-tuebingen.de/services/nilse/books/GnuplotinAction.pdf.

1.2 How does gnuplotClass work?

The class writes data and gnuplot code during run time to text files and issues system commands to compile these files and display output on screen or write as pdf file using gnuplot.

1.3 What is gnuplotClass good for

GnuplotClass is designed to produce final output during run time. It is generally not suitable to handle a very large number of plots.

1.4 Is gnuplotClass cross-platform compatible?

GnuplotClass works with any FORTRAN compiler/platform up to the point of issuing system level commands that compile gnu code. Unfortunately, issuing system commands is compiler and platform specific. This particular implementation of the class assumes Intel Fortran Compiler (Composer XE 2017 or later) and Windows operating system. It will not function on any other platform or even older Intel Fortran compilers. However, even in such a case, the class can be used by setting rungnu=.false. and commenting lines of code that issue commands to the system (run_gnuplotxxxx subroutines). In this mode gnuplotClass only generates files for later manual compilation. Alternatively, if system commands are supported by the compiler but they invoked differently, the class can be easily modified to obtain full functionality by changing the lines of code in run_gnuplotxxxx subroutines that issue system commands.

1.5 What do I need to add gnuplotClass to my code?

GnuplotClass comprises two modules: GnuDataSeriesClass and GnuFigureMakerClass. They can both be found in GnuPlotClass.f90 file. GnuFigureMakerClass must be added to the program. GnuDataSeriesClass is added automatically one GnuFigureMakerClass is added. The following illustrates how to add the class to the main program:

```
program test
use GnuFigureMakerClass
implicit none
type(GnuFigureMaker) :: figure
type(GnuDataSeries)  :: GnuSeriesY(1:4), GnuSeriesX

end program
```

The above code adds both modules and defines two objects that it operates on: the figure class (figure) and two data wrappers: GnuSeriesY(1:4) and GnuSeriesX. The figure class defines the figure environment for the gnuplot and data wrappers store user provided data for plotting and series specific formatting information passed to gnuplot.

gnuplotClass file must be added to the FORTRAN project. The downloaded gnuplotClass package contains a simple f90 program (test.f90) that demonstrates its functionality. This documentation also includes several complete examples at the end. Output files that gnuplotClass produces are stored in the current directory.

(!) IMPORTANT: Gnuplot must be installed and accessible from the command-line; i.e., the path to its folder must be known to the system. Do not forget to check path option when installing gnuplot.

2 gnuplotClass figure wrapper

GnuFigureMaker figure wrapper class must be defined by a declaration made in main program prior to calling plotting routines. It is a wrapper class that contains figure formatting information passed to gnuplot. To define the wrapper, use syntax:

```
type(GnuFigureMaker) :: figure
```

The GnuFigureMaker comprises of attributes listed below. Note that these attributes are private and cannot be accessed from the main program other than by using an appropriate method listed below (see after 'contains').

```
type :: GnuFigureMaker

integer, private             :: dimension_= 2
character(len=80), private   :: title_= ""
```

```
character(len=30), private :: xlabel_= "x"
character(len=30), private :: ylabel_= "y"
character(len=30), private :: zlabel_= "z"
character(len=30), private :: datefmt_= "%Y"

integer, private           :: xaxismode_=1
logical, private           :: grids_=.true.
logical, private           :: legend_=.true.

real*8, private            :: uppermarginy_=0.
real*8, private            :: lowermarginy_=0.
real*8, private            :: uppermarginz_=0.
real*8, private            :: lowermarginz_=0.
real*8, private            :: uppermarginx_=0.
real*8, private            :: lowermarginx_=0.

type(LabelType), allocatable, private    :: labels_(:)
integer, private :: numlabels_=0
type(ArrowType), allocatable, private    :: arrows_(:)
integer, private  :: numarrows_=0
type(CommandType), allocatable, private  :: commands_(:)
integer, private :: numcommands_=0

contains

procedure :: setTitle, clearTitle
procedure :: setDimension
procedure :: setAxisLabels, clearAxisLabels
procedure :: setTimeFormat, clearTimeFormat
procedure :: setMargins, clearMargins
procedure :: setGrids, clearGrids
procedure :: setLegend, clearLegend
procedure :: addLabel, clearLabels
procedure :: addArrow, clearArrows
procedure :: addCommand, clearCommands
procedure :: gnuplot2D
procedure :: gnuplot2D_multi
procedure :: gnuplot3D_auto, gnuplot3D_auto_2
procedure :: gnuplot3D, gnuplot3D_2
procedure :: gnuhistogram
```

```
end type GnuFigureMaker
```

The following is an example that defines a new figure class suitable for two-dimensional plotting. The first lines of code (1-3) specify the title of the figure and axis formatting / labeling. The following lines of code (lines 5-13) add two custom labels (text) (label1, label2) at user specified coordinates (xpos, ypos) and specify the angle (rotation) at which they will be displayed. The next few lines (lines 13-16) add an arrow going from position (from_epos, from_ypos) to position (to_xpos, to_ypos). These objects can be used to label the plot instead of the automatically generated label by gnuplot.

(!) Note that all attributes are automatically initiated at compile time. Some attributes are optional and when explicit specification is omitted these values determine the program's behavior.

```
call figure % setTitle(title = 'FigureTitle')
call figure % setTimeFormat(fmt="%Y")
call figure % setAxisLabels(xaxis='year', &
                            yaxis='temperature')
call figure % addLabel(label = 'label1', &
                       xpos=epoch(2000), &
                       ypos=5.0d0, &
                       rotation=0)
call figure % addLabel(label ='label2', &
                       xpos=epoch(2002), &
                       ypos=1.0d0, &
                       rotation=30)
call figure % addArrow(from_xpos=epoch(2000),  &
                       from_ypos=0.45d0, &
                       to_xpos=epoch(2000), &
                       to_ypos=1.93d0)
```

The following is the complete list of attributes of the gnuFigureMaker class and the associated methods:

1. *dimension_* specifies whether the figure will be used to display a 2D plot or a 3D plot. To modify this attribute from the main program code, use the following syntax:

   ```
   call figure % setDimension(dim=2) !for 2D plot
   call figure % setDimension(dim=3) !for 3D plot
   ```

2. *title_* attribute specifies the title of the figure. The title will appear at the top of the figure. To modify this attribute from the main program code, use the following syntax:

```
call figure % setTitle(title='mytitle')
!title is of character* type
```

3. *xlabel_*, *xlabel_* attributes specifies axis labeling. For 3D plots zlabel should be analogously added. To modify this attribute from the main program code, use the following syntax:

```
call figure % setAxisLabels(xaxis='my-oxaxis-label', &
                                    yaxis='my-oxaxis-label')
!xasis and yaxis are of character* type
```

4. *xaxismode_* is an automatic and hidden attribute that defines formatting of the ox-axis. *xaxismode_=2* means ox data is a time axis. Time data is internally stored as Unix time [Unix time measures the number of second since year 1970.] Refer to later section "A note on plotting time series data" how to enter time data and plot time series. To modify this attribute from the main program code, use the following syntax:

```
call figure % setTimeFormat(fmt="%m/%d/%y" )
!fmt string as define gnuplot documentation
```

   Defining time formatting automatically switches xaxismode to 2. Use to remove remove formatting information.

```
call figure % clearTimeFormat()
```

5. *grids_* attribute adds grids to the figure. To add or clear grids, use:

```
call figure % setGrids() !adds grids (default)
```

   and to remove grids use syntax

```
call figure % clearGrids() !clears grids
```

6. *legend_* specifies whether a legend should appear on the figure. To add or clear legend, use syntax:

```
call figure % setLegend() !adds legend (default)
```

and to remove grids use

```
call figure % clearLegend() !clears legend
```

7. *uppermargin*_* (where *=x, y, or z) are optional attributes that add additional margins to the figure's range along * axis. That is, if a series that ranges from -5 to 5 is plotted, margin can some additional space on each side of range so that -5-lowermarginy, 5+uppermarginy range appears in the figure. To change this attribute for 2D plots, use syntax:

```
call figure % setMargins(downy=23.d0, & !y-axis margin (2D)
                         upy=13.d0 )
!where downy, upy etc. are double precision  coordinates
```

   For 3D plots, use

```
call figure % setMargins(downy=3.d0, & !y axis margin
                         upy=3.d0,   &
                         downz=3.d0, & !z- axis margin
                         upz=3.d0    &
                         downx=3.d0, & !x- axis margin
                         upx=3.d0)
```

8. *labels_*, *arrows_* attributes contain a set of labels and arrows that are added to the figure at a user specified coordinates (and angle in case of labels). Position variables are of double precision and specify the beginning of the object and in case of arrows also the end of the arrow. Use the following syntax to add arrow and labels to the figure:

```
call figure % addArrow(from_xpos=1.0d0, &
                       from_ypos=1.0d0, &
                       to_xpos=1.1d0, &
                       to_ypos=1.1d0, &
                       color='#000000')
!where position variables from_. to_. are of double precision
!color is optional HEX rgb code
```

   and analogously for *addLabel_*:

```
call figure % addLabel(label='myLabelText', &
                       xpos=2.0d0,  &
                       ypos=1.0d0, &
```

```
                                rotation=45,
                                color='#000000')
!position variables are of double precision
!rotation is integer
!color is optional HEX rgb code
```

For 3D figures use *addArrow3D* and *addLabel3D* constructors that allow to enter the third coordinate as *zpos*. To clear labels and arrows use method clearArrows() and clearLabels().

In addition to the above attributes one can add any command to gnu code using addGnuCommand method. This method works similarly as addLabel method but only contains a single string attribute 'command.' For example, to redefine line 1 style, we would use:

```
call figure % addGnuCommand(command='set style line 1 &
                                        lt 2 lw 2 pt 3 ps 0.5' )
```

To clear all previously defined commands, enter:

```
call figure % clearGnuCommands()
```

3 gnuplotClass data wrapper

GnuDataSeries must be defined by a declaration made in main program prior to calling plotting routines. It is a data wrapper class that contains user defined data for plotting together and series-specific formatting information passed to gnuplot. To define the wrapper, use syntax:

```
type(GnuDataSeries) :: GnuSeriesY(1:4), GnuSeriesX
```

Data for plotting must ALWAYS be defined as array, even if it is of size 1 (i.e., define GnuSeriesY(1)). Data for ox axis must be defined as a scalar.

The above declaration creates two data wrappers: a vector GnuSeriesY of dimension (1:4) and a scalar GnuSeriesX. GnuSeriesY contains 4 series of data for plotting and GnuSeriesX contains data for the ox-axis. The attributes of GnuDataSeries wrapper are public and can be directly accessed from the main program. Use provided constructors to upload the data.

```
type GnuDataSeries
    character(len=128)    :: title      = ''
```

```
    character(len=128)    :: unit       = ''
    integer               :: dataType   = DATA_UNDEFINED_TYPE
    integer               :: offset     = 0
    real*8, allocatable   :: doubleValues(:)
    real*4, allocatable   :: singleValues(:)
    integer, allocatable  :: integerValues(:)
    real*8, allocatable   :: doubleValues2(:,:)
    real*4, allocatable   :: singleValues2(:,:)
    integer, allocatable  :: integerValues2(:,:)
    real*8, allocatable   :: timeValues(:)
    logical               :: userSuppliedStyle=.false.
    character(len=1)      :: linestyle='-'
    character(len=8)      :: linecolor='#3D59AB'
    character(len=1)      :: linewidth='3'
    integer               :: size
    integer               :: size2
end type GnuDataSeries
```

4 Uploading data for plotting

Here is an example that uploads data for plotting using GnuDataSeries object using an appropriate constructor syntax. The first line of the constructor defines GnuDataSeries object that stores the data, the second line specifies the title of the series, and the third line uploads data using a FORTRAN array (x(1:n,1:3), y1(1:n), y2(1:n), respectively). In this case x is time variable and a special constructor *constructTSeries* is used in this case. It is an integer array of dimension 1:n,1:3. See Section "A note on plotting time series data" for more details how to enter time data and use this constructor. The other attributes are optional and contain formatting information passed to gnuplot later. As detailed below, *linecolor* specifies hex rgb color of that gnuplot should use, *linewidth* specifies linewidth as string integer 'n' and *linestyle* specifies whether the series should be plotted as a line '-', a line with points ':' or with just points '.'.

```
call constructTSeries(DataSeries = GnuSeriesX, &
                      title      = 'time',      &
                      series     = x            )

call constructSeries(DataSeries = GnuSeriesY(1), &
```

```
                        title     = 'series1',        &
                        series    = y1,               &
                        linecolor= colorBankR(1),     &
                        linewidth='3',                &
                        linestyle='-',                &
                        offset    =0                  )

call constructSeries(DataSeries = GnuSeriesY(2), &
                        title     = 'series2',        &
                        series    = y2,               &
                        linecolor='#666666',          &
                        linewidth='3',                &
                        linestyle='-',                &
                        offset    =0                  )

call constructSeries(DataSeries = GnuSeriesY(3), &
                        title     = 'series3',        &
                        series    = y3,               &
                        linecolor='#D95F02',          &
                        linewidth='3',                &
                        linestyle='-',                &
                        offset    =0                  )

call constructSeries(DataSeries = GnuSeriesY(4), &
                        title     = 'series4',        &
                        series    = y4,               &
                        linecolor='#1B9E77',          &
                        linewidth='3',                &
                        linestyle='-',                &
                        offset    =0                  )
```

The following is the complete list of attributes that can be specified within the constructor:

1. *DataSeries* is the GnuDataSeries object that stores the data. The object must be previously defined using a appropriate declaration.

2. *title* is a string label.

3. *unit* is an optional string attribute that specifies individual axis labeling for this series. It is only useful in plotting data on sub-figures, in which

case axis are labeled according to each series label rather than the figure attribute. See Section on 2D plotting using sub-figures for more details.

4. *series* is a FORTRAN array with a vector of data or a matrix for plotting. For 2D plotting use FORTRAN arrays dimension (1:n) of type single, double or integer. For time axis, as for x above, a special integer array array of dimensions (1:n,1:7) (n is number of observations) must be used, where x(:,1)=year, x(:,2)=month, x(:,3)=day etc. This vector can also be abbreviated and instead provide only date information x(1:n, 1:3). See more details in Section "A note on plotting time series data." For 3D plotting use square arrays of dimension(1:n,1:n) and type single, double or integer. 3D plots do not take time data.

```
integer   :: x(nx,1:8)
real*8    :: y1(n1), y2(n2)
!nx is size of data vector x
!n1 is size of data vector y1, n2 y2 etc...
```

[basicstyle= , style=myCustomStyle] The following is an example of input arrays that would be consistent with the calls above. x generates dates for ploting and y1 and y2 contain sample data.

```
x=0
do  i=1,nx !time data specifying year, month and day
  x(i,1)=(1997+i); x(i,2)=1;  x(i,3)=1
end do

do  i=1,n1 !single or double specifying year, month and day
  y1(i)=6.0*i/n1
end do

do  i=1,n2 !time data specifying year, month and day
  y2(i)=5.0*i/n2
end do
```

5. *dataType* is automatically generated by the constructor and hidden from the user. It specifies data type associated with the input array. If the supplied data vector is of integer type DATA_INTEGER_TYPE is returned by the constructor; if data vector is of real*4 type DATA_SINGLE_PRECISION_TYPE is returned by the constructor; if DATA_DOUBLE_PRECISION_TYPE; if data vector contains date and time values DATA_TIME_TYPE is returned. See second "A note on plotting time series data" at the end.

6. *linecolor* is an optional string attribute that specifies HEX rgb color of the series that is passed to gnuplot as it is plotted. Refer to gnuplot documentation for details.

7. *linewidth* is an optional string attribute that specifies HEX rgb color of the series that is passed to gnuplot as it is plotted. Refer to gnuplot documentation for details.

8. *linestyle* is an optional string attribute that specifies whether gnuplot should plot using lines, '-', lines and points, ':', or just points, '.'.

9. *offset* is an optional integer attribute that specifies when given data begins with respect to the ox-axis data. For instance, by specifying offset=2, we tell gnuplot to forgo the first two observations and plot the given series starting from third value on the ox axis. Note that the ox axis data must be sufficiently long to accommodate extended size of the data that positive offset implies.

10. *size* and *size2* attributes are automatically generated and hidden from the user. They contain input array sizes along each dimension.

(!) IMPORTANT: Note that derived type allows series to be of different length, even though they are pooled together into a single object. However, the ox axis must be the upper envelope of all sizes to enable labeling of all data. Be careful with setting offset>0, as it makes data series effectively longer.

(!) IMPORTANT: If optional SERIES style specification IS omitted automatic format is used instead. This can be handy when color=.false. for the pdf mode. gnuplot will then use lines that are dotted in varying pattern so that they are easily distinguishable. Refer to compile time parameters of gnuplotClass discussed at the end.

5 2D plots

Once figure and data objects have been defined, gnuplotClass is ready for plotting.

To plot a single or multiple series of data in two dimensions on a single figure use the method gnuplot2D. The method operates on gnuFigureMaker object and takes dimension (1:n) GnuDataSeries vector for the oy-axis data

and a scalar GnuDataSeries object for the ox-axis data. For example, to plot a 2D graph of y1 and y2 defined previously with x on the ox output and output being written to a pdf file gnumy-fig.pdf (or gnu-cairo-my-fig.pdf depending on settings), use the following command:

```
call figure % gnuplot ( DataX=GnuSeriesX , &
                        DataY=GnuSeriesY(1:2), &
                        terminal='pdf', &
                        label='my-fig' )
```

[basicstyle= , style=myCustomStyle] To display the same graph on screen, use:

```
call figure % gnuplot ( DataX=GnuSeriesX , &
                        DataY=GnuSeriesY(1:2), &
                        terminal='screen', &
                        label='my-fig' )
```

DataX feeds ox-axis data. *DataY* feeds oy-axis data to be plotted against *DataX* and provides formatting information how each series should be plotted. *DataY* MUST BE a dimension(1:n) vector (where n=1 for plotting a single series) and DataX MUST be a scalar. Note that the attribute *label* is a string that is added to the files that are generated by gnuplotClass as output (i.e., the file containing data , tex file, if any, gnuplot command file, and output files). For example, in this case final output will written to file gnumy-fig.pdf, where 'my-fig' part of it comes from the label attribute. gnuplotClass by default retains all interim files with data and gnu code used to generate the plot. gnucom[label].txt contains the code, gnu-dat[label].txt contains data (!) IMPORTANT: In pdfcairo mode do not use label

attribute that contains characters that LaTex would classify as requiring math mode. For example, avoid underscores '_'. Cairo will return an error otherwise.

(!) IMPORTANT: pdfcairo mode requires pdflatex be present and accessible from the command line.

5.1 Example of a 2D plot

Here is a complete code that generates a 2D plot, writes it to a pdf file and displays on screen. The output of this code is illustrated in Figure 2.

```
program test
use GnuFigureMakerClass
type(GnuFigureMaker) :: fig0
type(GnuDataSeries)  :: GnuSeriesY(1:4), GnuSeriesX
```

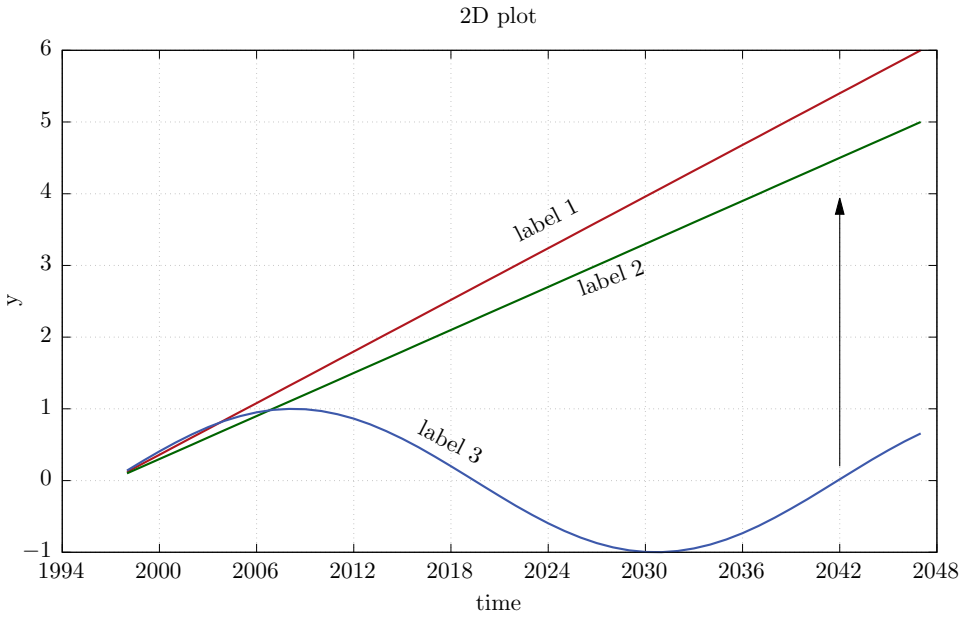

Figure 2: A sample 2D plot of multiple series on a single figure.

```
integer , parameter    :: n1=50, nx=50
integer :: x(nx,1:8)
real*8  :: y1(n1), y2(n1), y3(n1), y4(n1)
integer :: i

x=0
do   i=1,n1
     x(i,1)=(1997+i)
     x(i,2)=1
     x(i,3)=1
end do

do   i=1,n1
     y1(i)=6.0*i/n1
end do

do   i=1,n1
     y2(i)=5.0*i/n1
end do
```

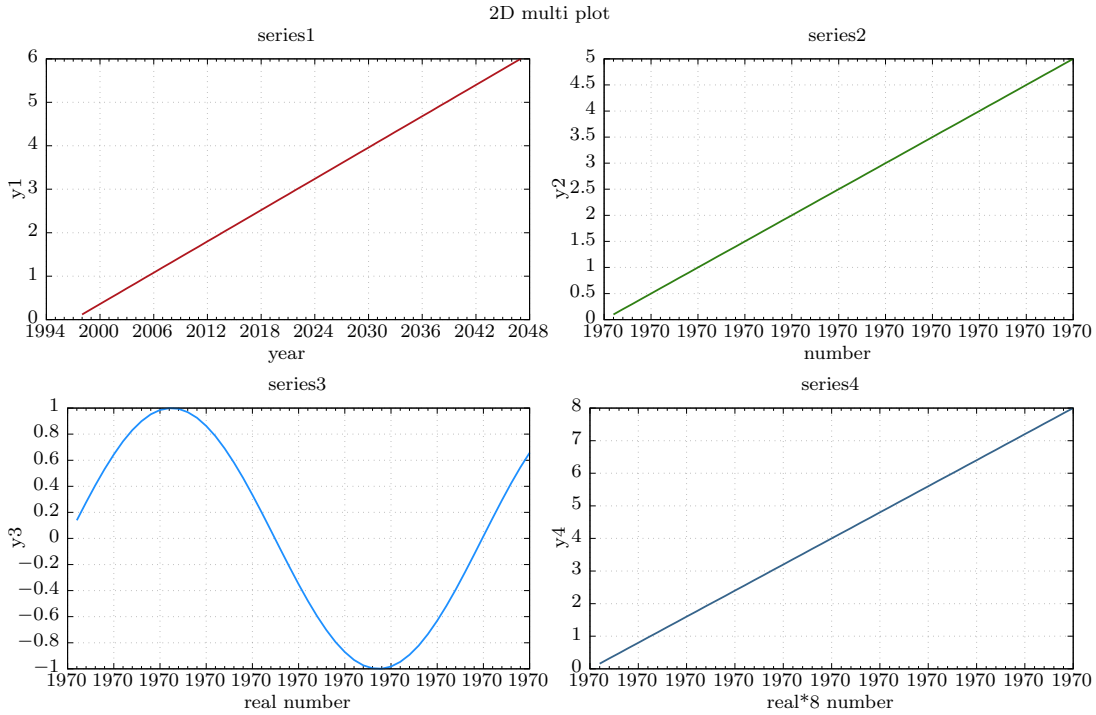

Figure 3: A sample 2D plot of multiple series split to sub-figures.

```
do   i=1,n1
     y3(i)=sin(7.0*i/n1)
end do

do   i=1,n1
     y4(i)=8.0*i/n1
end do

!set figure environment
!title, date format, and axis labeling
call fig0 % setTitle(title = '2D plot')
call fig0 % setTimeFormat(fmt="%Y")
call fig0 % setAxisLabels(xaxis='time', yaxis='y')

!add custom labels
call fig0 % addLabel(label = 'label 1', &
```

```
                        xpos=epoch(2024),  &
                        ypos=3.6d0,        &
                        rotation=27         )

call fig0 % addLabel(label = 'label 2', &
                        xpos=epoch(2028),  &
                        ypos=2.8d0,        &
                        rotation=20         )

call fig0 % addLabel(label = 'label 3', &
                        xpos=epoch(2018),  &
                        ypos=0.6d0,        &
                        rotation=-30        )

!add an arrow
call fig0 % addArrow(from_xpos=epoch(2042),&
                        from_ypos=0.2d0,       &
                        to_xpos=epoch(2042),   &
                        to_ypos=3.94d0          )

!clear automatic labels
call fig0 % clearLegend()

!ox data
call constructTSeries(DataSeries = GnuSeriesX,     &
                       title     = 'calendar time', &
                       series    = x                 )

call constructSeries(DataSeries = GnuSeriesY(1),&
                        title     = 'series1',       &
                        series    = y1,              &
                        linecolor= colorBankR(1)))

call constructSeries(DataSeries = GnuSeriesY(2), &
                        title     = 'series2',       &
                        series    = y2,              &
                        linecolor= colorBankG(1)     )

call constructSeries(DataSeries = GnuSeriesY(3),  &
                        title     = 'series3',       &
```

```
                    series    = y3,                 &
                    linecolor= colorBankB(1)       )

!set compile time option to use cairo terminal to generate pdfs
pdfcairo = .true.

!plot all 3 series
call fig0 % gnuplot2D(  DataX=GnuSeriesX,        &
                        DataY=GnuSeriesY(1:3), &
                        terminal='pdf',          &
                        label='myfig_2D'         )

call fig0 % gnuplot2D(  DataX=GnuSeriesX,        &
                        DataY=GnuSeriesY(1:3), &
                        terminal='screen',       &
                        label='myfig_2D'         )

!plot again but just the first series
call fig0 % gnuplot2D(  DataX=GnuSeriesX,        &
                        DataY=GnuSeriesY(1),   &
                        terminal='pdf',          &
                        label='myfig_2D'         )

call fig0 % gnuplot2D(  DataX=GnuSeriesX,      &
                        DataY=GnuSeriesY(1), &
                        terminal='screen',     &
                        label='myfig_2D'       )

end program
```

To plot multiple series of data in two dimensions on multiple sub-figures use the method gnuplot2D_multi. Similarly, the method takes dimension (1:n) GnuDataSeries vector for the oy-axis and a scalar GnuDataSeries object for the ox axis. For example, to plot a 2D graph of x, y1 and x, y1 defined previously and output on single window with a 2-by-2 arragement of subfigures, use syntax:

```
call figure % gnuplot2D_multi(  fontsize=8,                &
                                numrows=2,                 &
                                numcols=2,                 &
                                DataX=GnuSeriesXX(1:4), &
                                DataY=GnuSeriesY(1:4),  &
```

```
                                    terminal='pdf',          &
                                    label='myfig_multi'      )
```

The additional attributes specify the fontsize, the number of rows of subfigures and the number of columns of subfigures. Figure 3 illustrates a sample plot using this method.

(!) IMPORTANT: In this mode gnuplotClass uses *unit* attribute of GnuDataSeries to label the axis of each subplot.

6 3D plots

3D plots require a data object with a matrix as an input. In addition, data for ox- and oy-axis can be separately provided. The remaining attributes for gnuplot work analogously. For example, the following code uses data constructor to upload data for 3D plotting:

```
!generate data
 do   i=1,n1
        do j=1,n1
            x3D(i)=real(i)/10.
            y3D(j)=real(j)/10.
            xyz2(i,j)=cos(real(i)*real(j)/(200.) ) !3d matrix format
    end do
end do

!define figure
call fig2 % setTitle(title = 'My first GNU 3D plot')
call fig2 % setAxisLabels(xaxis='x', yaxis='y', zaxis='z')
call fig2 % setDimension(dim=3)

!upload ox, oy, and oz data
call constructSeries(DataSeries = GnuSeriesX3D, &
                                  title     = "3d",    &
                                  series    = x3D    )

call constructSeries(DataSeries = GnuSeriesY3D,  &
                                   title     = "3d",    &
                                   series    = y3D    )

call constructSeries(DataSeries = GnuSeriesXYZ, &
```

```
                                    title     = "3d",     &
                                    series    = xyz2    )
```

The class can plot up to two oz-matrices on a single figure.oy-axis data are provided, use the syntax

```
call figure % gnuplot3D(DataX=GnuSeriesX3D,   &
                        DataY=GnuSeriesY3D,   &
                        DataXYZ=GnuSeriesXYZ, &
                        terminal='pdf',   &
                        label='myfig'     )
```

and if ox- and oy-axis data is not supplied, use syntax

```
call figure % gnuplot3D_auto(DataXYZ=GnuSeriesXYZ,   &
                             terminal='pdf',   &
                             label='myfig'   )
```

gnuplotClass can also plot two matrices on the same figure. To do that, use syntax:

```
call figure % gnuplot3D_2(DataX=GnuSeriesX3D, &
                          DataY=GnuSeriesY3D,     &
                          DataXYZ=GnuSeriesXYZ,   &
                          DataXYZ1=GnuSeriesXYZ1,   &
                          terminal='pdf', &
                          label='myfig'   )
```

In this case both GnuSeriesXYZ and GnuSeriesXYZ1 are plotted in the same figure.

(!) IMPORTANT: Offset is inactive in 3D mode. Data sizes must fully agree in all dimensions. Time data format is not compatible with 3D plots.

6.1 Example of a 3D plot

Here is a complete code that generates a 3D plot, writes it to a pdf file and displays it on the screen. The output of this code is illustrated in Figure 4 and 5.

```
program test
use GnuFigureMakerClass
implicit none

type(GnuFigureMaker) :: fig2
```

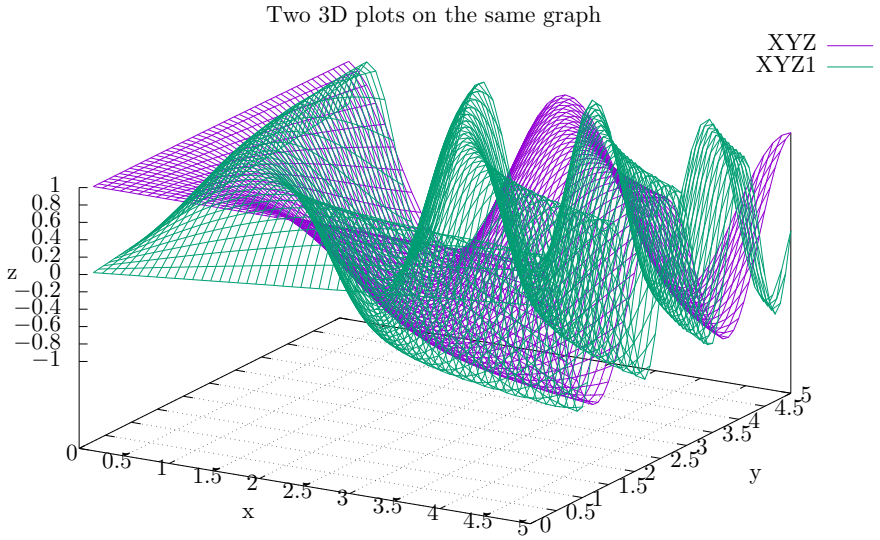

Figure 4: A sample 3D plot of two matrices.

```
type(GnuDataSeries) ::   GnuSeriesXYZ,GnuSeriesXYZ1
type(GnuDataSeries) ::   GnuSeriesX3D, GnuSeriesY3D

integer, parameter   :: n1=50, nx=50

real*8          :: x3D(n1), y3D(n1)
real*8          :: xyz2(n1,n1), xyz3(n1,n1)

!generate data
do   i=1,n1
        do j=1,n1
            x3D(i)=real(i)/10. !ox
            y3D(j)=real(j)/10. !oy

            xyz2(i,j)=cos( real(i)*real(j)/200. ) !matrix  plot 1
            xyz3(i,j)=sin( real(i)*real(j)/100. ) !matrix  plot 2
    end do
end do
```

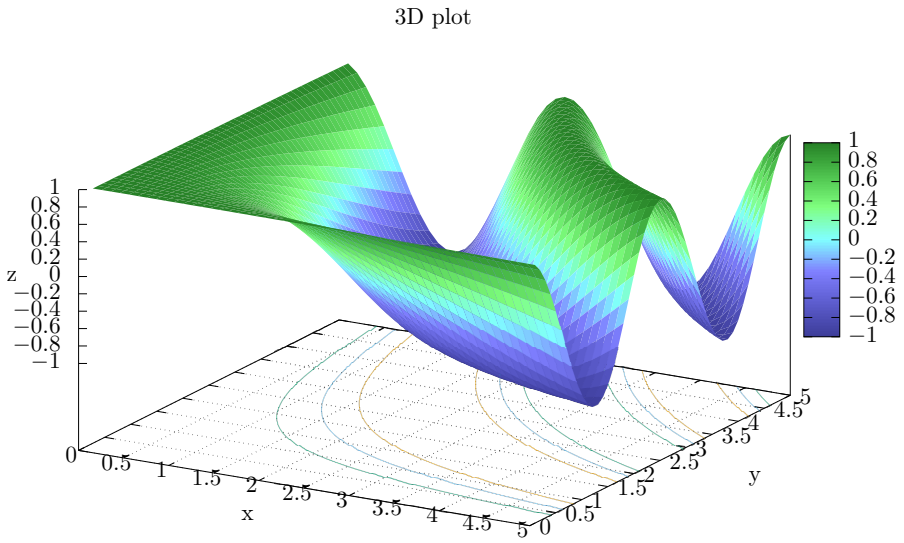

Figure 5: A sample 3D plot of a single matrix.

```
!set figure environment
call fig2 % setTitle(title = '3D plot')
call fig2 % setAxisLabels(xaxis='x', yaxis='y', zaxis='z')
call fig2 % setDimension(dim=3)

!upload data series for plotting
call constructSeries(DataSeries = GnuSeriesX3D,     &
                     title      = "X",              &
                     series     = x3D    )

call constructSeries(DataSeries = GnuSeriesY3D,     &
                     title      = "Y",              &
                     series     = y3D    )

call constructSeries(DataSeries = GnuSeriesXYZ,     &
                         title  = "XYZ",            &
                         series = xyz2    )
```

```
pm3d=.false. !set grid filling to off to improve visibility

!plot 2 matrices of data to pdf and on screen
call fig2 % gnuplot3D_2(DataX  = GnuSeriesX3D,     &
                        DataY  = GnuSeriesY3D,     &
                        DataXYZ= GnuSeriesXYZ,  &
                        andDataXYZ=GnuSeriesXYZ1, &
                        terminal='pdf',         &
                        label='myfig-3D-2'         )

call fig2 % gnuplot3D_2(DataX=GnuSeriesX3D,        &
                        DataY=GnuSeriesY3D,        &
                        DataXYZ=GnuSeriesXYZ,      &
                        andDataXYZ=GnuSeriesXYZ1, &
                        terminal='screen',         &
                        label='myfig-3D-2'         )



!same but just plot first matrix

pm3d=.true. !set grid filling to on to see the effect


call fig2 % gnuplot3D (DataX  = GnuSeriesX3D,     &
                       DataY  = GnuSeriesY3D,     &
                       DataXYZ= GnuSeriesXYZ,  &
                       terminal='pdf',         &
                       label='myfig-3D-2'         )

call fig2 % gnuplot3D(DataX=GnuSeriesX3D,        &
                      DataY=GnuSeriesY3D,        &
                      DataXYZ=GnuSeriesXYZ,      &
                      terminal='screen',         &
                      label='myfig-3D-2'         )
end program
```

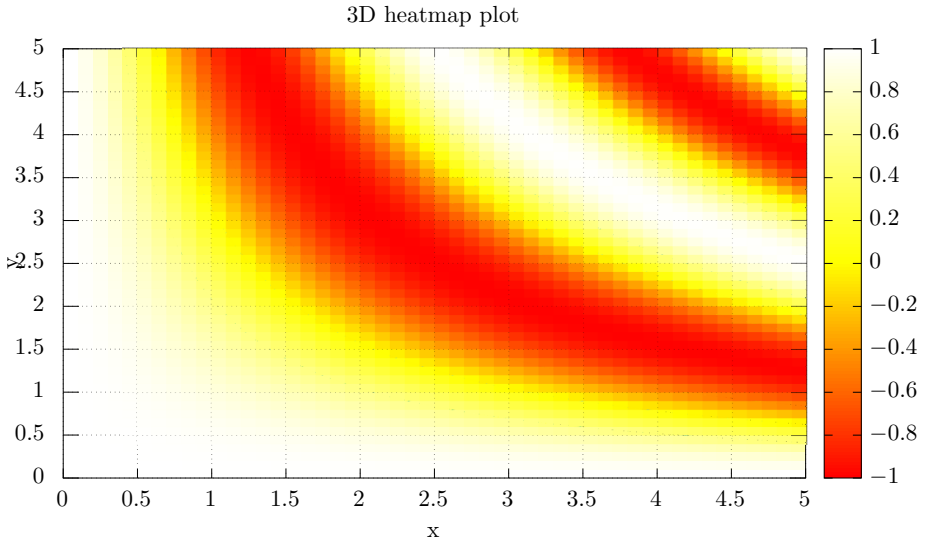

Figure 6: A sample heatmap plot.

7 Heatmap plots

To obtain a heat plot from any 3D plot add the following lines of code prior to the plot command:

```
call fig3 % addCommand('set view map') !change view to map plot
default_palette=heat_palette !change palette to heat palette
pm3d=.true. !switch to pm3d to fill in the grids
```

Figure 6 illustrates a sample plot using this method.

8 Histogram plots

Use *gnuplothistogram* method to obtain a histogram plot of a bunch of data. Histogram plot take a scalar *GnuDataSeries* object and counts the number of occurrences with each of n intervals defined between the minimum and maximum of the data series. Use the following call:

```
call figure % gnuhistogram(DataX=GnuSeriesRnd1,     &
                           numbars = 10,             &
                           terminal='pdf',           &
```

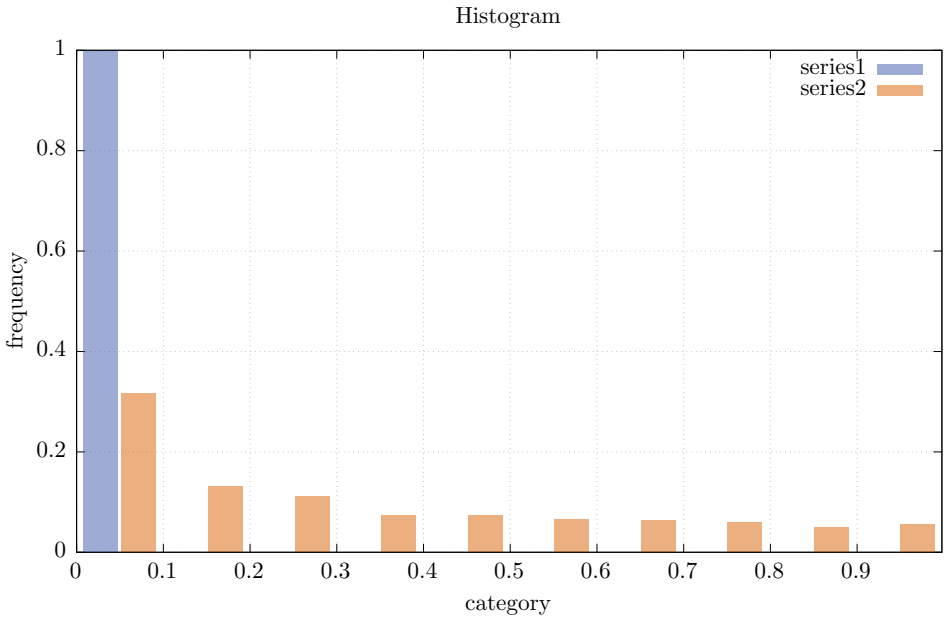

Figure 7: A sample histogram plot.

```
                                    label='myhist'                  )
```

GnuSeriesRnd1 is *GnuDataSeries* scalar object containing data, numbars defines the number of evenly spaced grid points between series minimum and maximum. (The current version of the class does not allow to specify unevenly spaced grids.)

9 Example of a histogram plot

Here is a complete code that generates a histogram plot, writes it to a pdf file and displays it on the screen. The output of this code is illustrated in Figure 7.

```
program test
use GnuFigureMakerClass
implicit none

type(GnuFigureMaker) :: fig4
type(GnuDataSeries) :: GnuSeriesRnd1(1:2), GnuSeriesRnd(1)

integer, parameter  :: n1=50
integer, parameter  :: nsq=n1*n1
```

```
real                    :: rmatrix1(nsq)
real                    :: rmatrix2(nsq)

integer, dimension(1) :: seed = (/3/)

!generate data
call random_seed (PUT=seed)
call random_number( rmatrix2 )
rmatrix2=rmatrix2*rmatrix2
rmatrix1=.05

!set figure environment
call fig4 % setTitle(title = 'Histogram')
call fig4 % setAxisLabels(xaxis='category', yaxis='frequency' )
call fig4 % setDimension(dim=2)

!upload data for plotting
call constructSeries(DataSeries = GnuSeriesRnd1(1), &
                     title      = "series1",        &
                     series     =  rmatrix1          )

call constructSeries(DataSeries = GnuSeriesRnd1(2), &
                     title      = "series2",        &
                     series     =  rmatrix2          )

!generate first histogram with 2 series of data
call fig4 % gnuhistogram(DataX=GnuSeriesRnd1,    &
                         numbars = 10,           &
                         terminal='pdf',         &
                         label='myhist'          )

call fig4 % gnuhistogram(DataX=GnuSeriesRnd1,    &
                         numbars = 10,           &
                         terminal='screen',      &
                         label='myhist'          )

!plot another one with just one series of data

call constructSeries(DataSeries = GnuSeriesRnd(1), &
                     title      = "series2",        &
```

```
                                series    =  rmatrix2          )

call fig4 % gnuhistogram(DataX=GnuSeriesRnd, &
                         numbars = 10,       &
                         terminal='pdf',     &
                         label='myhist0'     )

call fig4 % gnuhistogram(DataX=GnuSeriesRnd,     &
                         numbars = 10,           &
                         terminal='screen',      &
                         label='myhist0'         )
```

10 A note on plotting time series data

Gnuplot internally uses epoch time (Unix time). GnuplotClass constructor constructTSeries converts calendar time to epoch time prior to storing it in *GnuDataSeries* object. Full calendar time specification involves year, month, day, difference in minutes from the universal time, and then hour (0:24), minute and second.

To convert calendar time to epoch time, a special constructor constructTSeries must be used. The constructor takes a vector of integers of dimension(1:n,1:7), where n is the number of data points. Each entry 1 through 7 contains an integer denoting year, month, day etc... The constructor can also take an abbreviated vector (1:n,1:3) specifying date but not time. In such a case time is assumed to be 00:00:00 universal time. The 2D example above illustrates the use of time data.

To locate labels or arrows on the horizontal axis containing time data use epoch function available in gnuplotClass. This function converts calendar time to epoch time. Epoch function is of the format epoch(year, month, day, shift, hour, minute, second), where everything from month onward is optional. However, if any element of time vector is specified, it must be specified completely. See above code how this function has been used to locate labels and arrows.

11 Compile time settings

GnuplotClass is by default set to store files and use cairo pdf terminal. However, it can also silently write plots into pdf files and used basic pdf terminal instead of cairopdf terminal. It can also only generate files and forgo issuing system level commands to compile them. To change these option, see parameter settings in the GnuDataSeriesClass heading. These parameters can be modified from the main

code or when the are initialized within the class. Here is the list of some of main parameters:

```
logical :: rungnu=.true.
!if true FORTRAN issues system command to plot figures

logical :: pdfcairo = .true.
!if true uses cairo terminal for pdf plots

logical :: color = .true.
!if false true produces color figure in pdf cairo mode
!otherwise produces monochrome figure

logical, parameter :: stealthmode=.true.
!if true gnuplotClass leaves only final output files
! and erases all GNU codes and supporting files used
! to compile figures

!pm3d filling of grids
logical  :: pm3d=.false.
!if true uses pm3d mode to fill grids

!Default palette for 3D plotting
character(len=128) :: default_palette=luminance_palette
!default palette for 3D plotting and heat plotting

character(len=128) :: default_colorA=colorBankB(1)
character(len=128) :: default_colorB=colorBankG(1)
!default colors for histogram
```

If pdfcairo terminal is available on a given platform, gnuplotClass can use it to generate nicer pdfs by encapsulating a raw plot in a tex code that typesets all the labels using Latex compiler (pdflatex must be present). This results in a higher printout quality of mathematical expressions. stealthmode = .false. keeps all interim files that can later be used to recompile figures manually. Set to .true. to remove all interim files and leave just final output.

12 Issues and quick fixes

1. Previously generated files on same platforms do not allow pdflatex to generate new pdf files with the same name. Use call eraseGnuFile() to clean the

current folder from all files that start with gnu- and have .txt, tex or pdf extension.