GGGmanpage

D. COHEN[1,2]

July 27, 2006

[1] *Dept. de Mathématiques, Univ. de Genève, CH-1211 Genève 4, Switzerland.*
email: David.Cohen@math.unige.ch

[2] *Mathematisches Institut, Univ. Tübingen, D-72076 Tübingen, Germany.*
email: Cohen@na.uni-tuebingen.de

1 Content of the file GGGraphics.tar

The *GGGraphics.tar* file contains the following files

- *example-house.f*: a simple example,
- *GGG.f*: the GGG-routines,
- *GGGraphics.sty*: the GGG-package for LaTeX,
- *GGG-Routines.txt*: a brief description of the routines,
- *README*: you should read it . . . ,
- *tex-text.tex*: example for the corresponding LaTeX file,

and two directories

- *logo-example* containing *logo.f, logo.inp, logo.ps* and *README*: this draws the logo,
- *sophist-example* containing *adams4.f, ddeflat.f, horse_make.dat, horse_make.f* and *README*: the root locus curve of Adams4.

2 The structure of a file

To use the GGGraphics routine, your Fortran file (.f) should have the following form:

```
C     Initialization of the variables
C
C --------- SIZE OF THE PICTURES IN CM AND DISTANCES ------------
      COMMON/SIZES/SIZE(4)
      DATA SIZE/WIDTH,HEIGHT,LEFT-RIGHT_DIST,UP-DOWN_DIST/

C --------- INITIALIZATION GRAPHICS ------------------------------
C  The statement CALL BEGIN_GGG('name') initiates
C  the environment, which will produce a ps-file
C  'name.ps' and a LaTeX input file 'name.inp'
C  to be inserted as follows in the LaTeX file:
C  \begin{figure}[h]
```

```
C    \centering
C     \GGGinput{path}{name.inp}
C    \caption{Test for GGGnugnat}\label{Gnugnat}
C  \end{figure}
C  The character 'name' should be the name of your Fortran file

      CALL BEGIN_GGG('name')

C --------- MAKES A FRAME ---------------------------------------
      CALL FRAME

C --------- DRAW SOME GRAPHICS ----------------------------------
C  use of the routines in GGG.f

C --------- CORRECT BOUNDING BOX (LEFT,RIGHT,DOWN,UP; IN CM) ----
C  for fine adjustment of the picture inside the text

      CALL GREATER_BOUNDINGBOX(0.,0.,0.,0.)

C --------- AND THAT'S THE END ----------------------------------
C  writes the necessary macros and produces final files

      CALL END_GGG

      END
```

A description of the various subroutines will be given in the following. In Section 3, we will see how to initialize the environment to produce graphics with GGG. Section 4 will describe the subroutines used to draw graphics. The final section (Section 5) gives a small description of how to close the GGGraphics environment.

3 Initialization of the environment

In your FORTRAN 77 (90 or 95)file, just after the initialization of your variables, if you want to use the GGGraphics environment, you have to give the sizes of the pictures and the distance between them. This is done by adding the following lines in your code

```
COMMON/SIZES/SIZE(4)
      DATA SIZE/Width,Height,Left-Right_Dist,Up-Down_Dist/
```

here the sizes and distances are in centimeters. The variables *Width* and *Height* are the width, resp. the height, of your picture. *Left-Right_Dist* and *Up-Down_Dist* describe the left-right (resp. the up-down) distances between two pictures.

Then we have to initialize the graphic environment, for this we use

```
CALL BEGIN_GGG('name')
```

where *name* is the name of your file (omitting the extension .f). This will produce a ps-file *name.ps* and a LaTeX input file *name.inp* which has to be inserted as follows in the LaTeX file:

```
\begin{figure}[h]
  \centering
  \GGGinput{path}{name.inp}
  \caption{Test for GGGraphics}\label{fig:test}
\end{figure}
```

Do not forget to insert the package GGGRAPHICS.STY in your LaTeX file (using the standard LaTeX input command *usepackage*{*GGGraphics*}).

Finally to draw a frame simply type

```
CALL FRAME
```

in your code.

4 Description of the FORTRAN subroutines

In this section, we will describe the subroutines containing in the file *GGG.f*.

4.1 Placement

The following subroutines are used to place the pictures in your postscript file

1. MICKEY_MOUSE
2. MASOG
3. PAPER_CORNERS and XY_LIMITS

Let us begin with the subroutine

- MICKEY_MOUSE(I,J,XMIN,XMAX,YMIN,YMAX)

  This subroutine takes as argument two integers *I* and *J* and four real numbers *XMIN*, *XMAX, YMIN, YMAX*.

  It places a picture of size *SIZE(1),SIZE(2)* (in centimeters) on the postscript file like in a Mickey Mouse comics. The variables *SIZE(1),SIZE(2)* are defined in *COMMON/SIZES/SIZE(4)*, see above. This subroutine can be used several times.

  I,J are the positions like the indices of a matrix. Negative values for *I* fill the matrix from below. *XMIN,XMAX,YMIN,YMAX* are the bounds of the local coordinates.
  Let us illustrate this with an example. In Figure 1, we draw three pictures

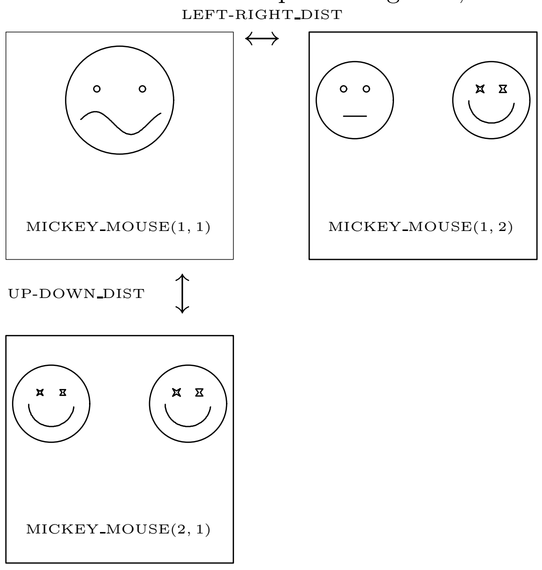

Figure 1: 3 pictures placed with MICKEY_MOUSE

and place them like in a comics book with

```
... initialization
C draw of the first picture
CALL MICKEY_MOUSE(1,1,XMIN,XMAX,YMIN,YMAX)
CALL FRAME
...
C draw of the second picture
CALL MICKEY_MOUSE(1,2,XMIN,XMAX,YMIN,YMAX)
CALL FRAME
...
C draw of the third picture
CALL MICKEY_MOUSE(2,1,XMIN,XMAX,YMIN,YMAX)
CALL FRAME
...
```

- MASOG(XMIN,XMAX,YMIN,YMAX)

  It takes as argument four real numbers *XMIN,XMAX,YMIN,YMAX*.

  Similar to the subroutine MYCKEY_MOUSE, but with automatic choice of *I,J*.

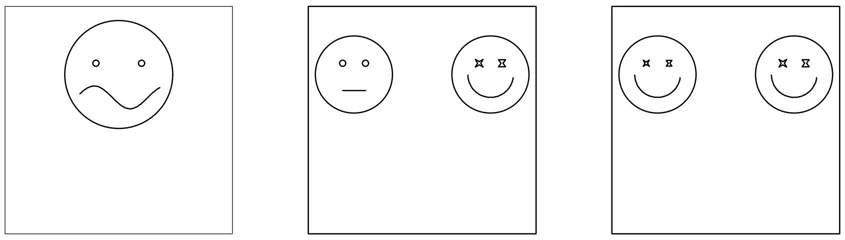
Figure 2: Same example as before, but with a call of MASOG

```
... 0
C draw of the first picture
CALL MASOG(XMIN,XMAX,YMIN,YMAX)
CALL FRAME
...
C draw of the second picture
CALL MASOG(XMIN,XMAX,YMIN,YMAX)
CALL FRAME
...
C draw of the third picture
CALL MASOG(XMIN,XMAX,YMIN,YMAX)
CALL FRAME
...
```

- PAPER_CORNERS(U1,V1,U2,V2) and XY_LIMITS(XMIN,XMAX,YMIN,YMAX)

  Both subroutines take as argument four real numbers.

  They are useful for drawing several pictures of different sizes: first define coordinates of (left-lower, right-upper) paper corners in *cm* in the subroutine PAPER_CORNERS, then *xy*-limits of local coordinates with XY_LIMITS. In Figure 3, we take the same pictures as in the first example and we resize the second and third one using the subroutine PAPER_CORNERS. To keep the proportions, we have to change the local coordinates using XY_LIMITS.

```
... initialization
CCC --- local coordinates
  XMIN=0.
  XMAX=1.
```

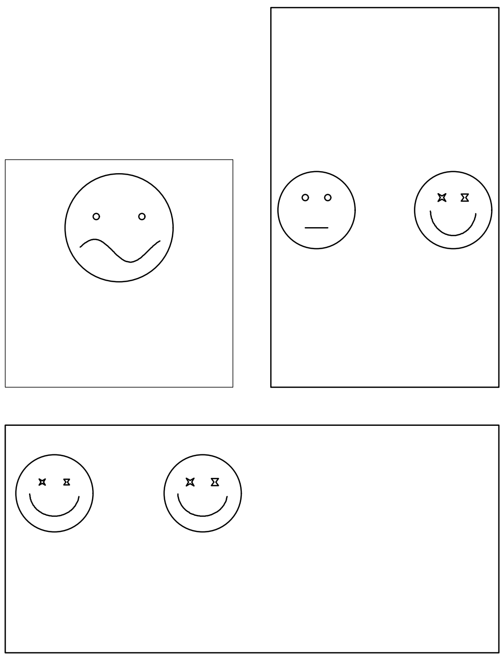
Figure 3: Same example as before, but with PAPER_CORNERS

```
  YMIN=0.
  YMAX=1.
CCC --- coordinates of the paper corners
  U1=0.
  V1=0.
  U2=3.
  V2=3.
CCC --- draw of the first picture
  CALL PAPER_CORNERS(U1,V1,U2,V2)
  CALL XY_LIMITS(XMIN,XMAX,YMIN,YMAX)
  CALL FRAME
...
```

Exercises

- *Use the subroutine* MICKEY_MOUSE *to draw four pictures. The third one should not have a frame. Vary the space between the pictures.*
- *Use the subroutine* PAPER_CORNERS *to draw two pictures. The first should have twice the width of the second.*

4.2 Line type, line thickness and character size

- SELECT_LINE_TYPE(ITYPE)

  It takes as argument an integer *ITYPE* ranging from 0 to 7.

  This subroutine selects the line type (see Figure 4 below).

  For *ITYPE*= 0, the subroutine draws dots at output points.

- THICK_PIXEL(ITHICK)

  It takes as argument an integer *ITHICK*.

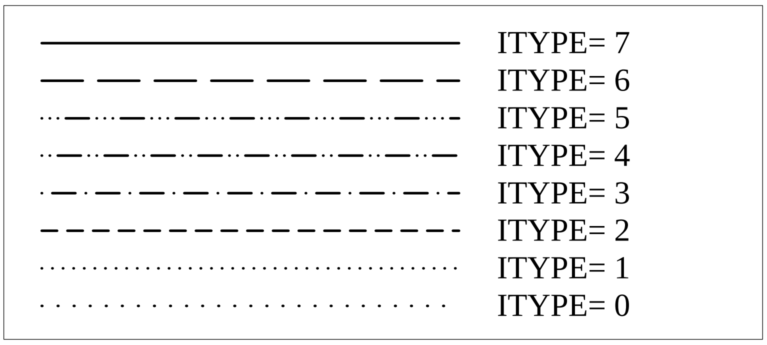

Figure 4: Different line types obtained with SELECT_LINE_TYPE

This subroutine sets the line thickness in pixels (standard is 2 or 4).

- SCALE_CHAR(SCALE)

  It takes as argument a real number *SCALE*.

  This subroutine sets the character size for the subroutine TEXT (see Subsection 4.7) and for the axes labeling (LABEL) (see Subsection 4.6). The standard size is 1.

4.3 Lines and curves

- LINE(XB,YB,XE,YE) and LINE_OPEN(XB,YB,XE,YE)

  These two subroutines take as argument four real numbers *XB,YB,XE,YE*.

  The subroutine LINE draws a line connecting the two points *XB,YB* and *XE,YE*. The subroutine LINE_OPEN also draw a line connecting these two points but without control of the drawing box. In Figure 5, we plot

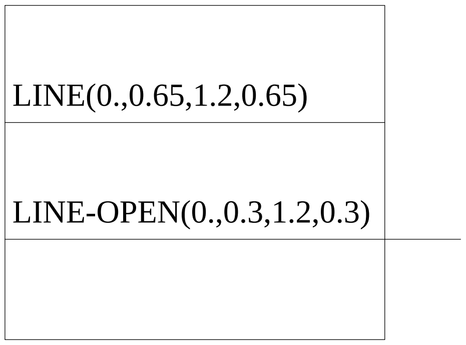

Figure 5: Two lines obtained with LINE an LINE_OPEN

two lines. The upper line is given by the subroutine LINE, the lower one by the subroutine LINE_OPEN. You can notice that the lower line goes outside the frame.

- LING(XDES,YDES,NDES) and LING_OPEN(XDES,YDES,NDES)
  These two subroutines take as argument two real vectors *XDES, YDES* and an integer *NDES* (the dimension of the two vectors).

  The subroutine LING connects *NDES* points with given coordinates *(XDES(I),YDES(I))*, for *I=1,NDES*. Similar to LINE_OPEN, use LING_OPEN if you want to draw outside the box.

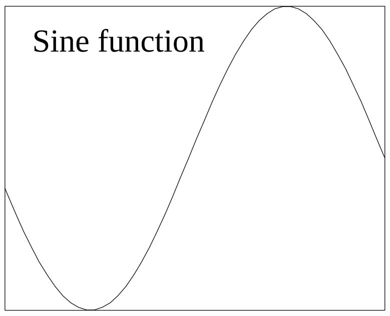

Figure 6: The sine function plotted with LING

The sine function of the Figure 6 is obtained with the help of the following code

```
...
NDES=50
DO I=1,NDES
   XDES(I)=-PI+I*(2.*PI)/NDES
   YDES(I)=SIN(XDES(I))
END DO
CALL LING(XDES,YDES,NDES)
...
```

Exercises

- *Draw two vertical lines. The first one should be a doted line, the second a continuous line. Vary the line types and thickness (see Subsection 4.2).*
- *Draw two pictures with a frame. Inside the first one, place a triangle and inside the second a circle.*

4.4 Arrows

- The simplest way to draw an arrow is to use the subroutine ARROW(XB,YB,XE,YE,AMM). It takes as argument, five real numbers *XB,YB,XE,YE,AMM*.

  This subroutine draws an arrow connecting the two points *(XB,YB)* and *(XE,YE)*, *AMM* indicating the size of the arrow tip. In Figure 7, we draw four arrows with tip size going from one to four.

- SHY_ARROW(XB,YB,XE,YE,AMM,PERCENT) takes as argument six real numbers. It draws an arrow but cuts off *(100-PERCENT)/2* of the length at both sides, if $|PERCENT| \leq 1$, it is replaced by *100*PERCENT*.

  In Figure 8, we draw four arrows using SHY_ARROW with various choice of the value of *PERCENT*.

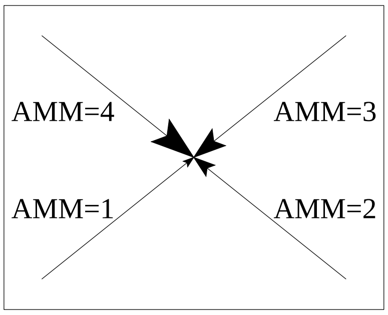

Figure 7: Arrows using the subroutine ARROW

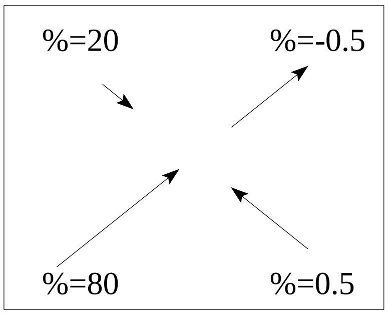

Figure 8: Arrows with various choice of *PERCENT* (the %)

- LINGARROW(XDES,YDES,NDES,AMM) takes as argument two vectors *XDES,YDES*, an integer *NDES* (the size of the vectors) and a real number *AMM*.

  This subroutine draws polygons with an arrow at the end (see Figure 9). It is for example useful for drawing flows.

- LINGARROWS(XDES,YDES,NDES,AMM,INTER)

  In addition to the preceding subroutine, LINGARROWS takes another input argument *INTER* (an integer). It draws arrows at distance *INTER* from the beginning to the end of the list of our points *XDES(I)*, *YDES(I)*.

  In Figure 9, we plot the graph of the sine function using the two subroutines LINGARROW and LINGARROWS.

- CONNECTICUT(I1,J1,I2,J2,AMM,PERCENT)

  It takes as argument four integers *I1,J1,I2,J2* and two real numbers *AMM,PRECENT*.

  This subroutine connects arrows with tip size *AMM* between boxes drawn by the subroutine Mickey_Mouse.

Exercises

- *Draw a circle with six arrows pointing at the center.*
- *Using* LINGARROWS*, draw a square which has an arrow at each corners.*

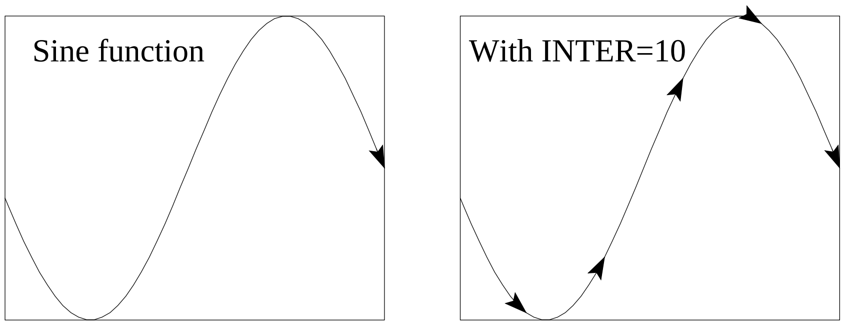

Figure 9: The sine function with LINGARROW and LINGARROWS

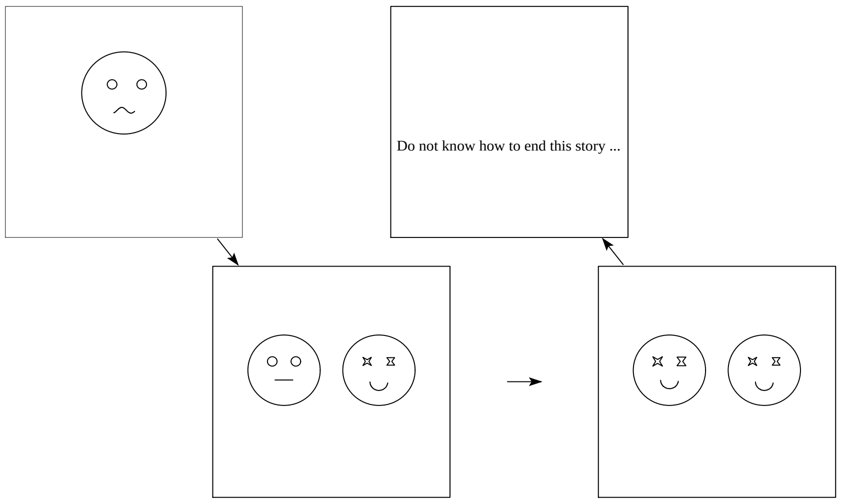

Figure 10: CONNECTICUT in action ...

4.5 Bubbles

- BUBBLE(XX,YY,AMMBIG,GREY,SYM).

  This subroutine takes as argument four real numbers *XX,YY,AMMBIG,GREY* and a string *SYM* which can be 'circle', 'square', 'losange', 'xstar', 'pstar', 'utri', 'dtri', 'umtri', 'dmtri', 'hardy', 'laurel', 'mpent', 'mhex', 'pent', 'hex', or 'mmhex' (see Figure 11).

  This subroutine draws the symbol *SYM* at position *(XX,YY)*. The size of the symbol is given by *AMMBIG* (in mm) and its brightness by *GREY* (between 0 and 1).

  In Figure 11, we draw all of the possible choices of the value of *SYM*.

- BUBBLE_OPEN(XX,YY,AMMBIG,GREY,SYM)

  Same as the previous subroutine but without control of the drawing box.

- RGB_BUBBLE(XX,YY,AMMBIG,RED,GREEN,BLUE,SYM)

  Same arguments as for the subroutine BUBBLE except that *GREY* is replaced by three real numbers *RED*, *GREEN*, *BLUE* (between 0 and 1).

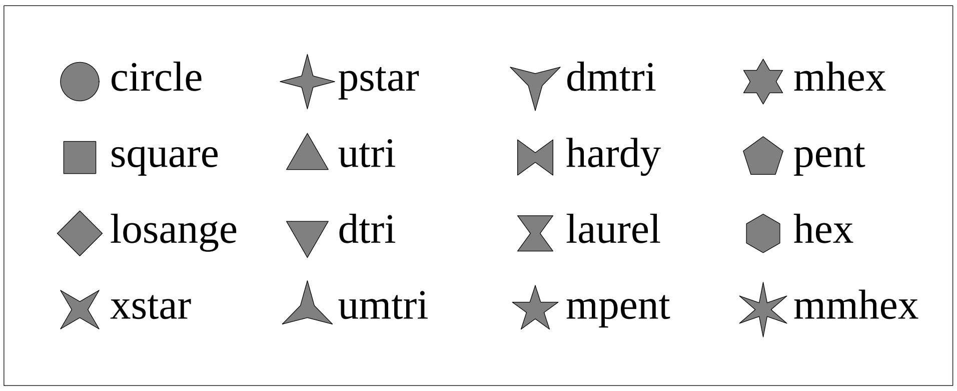

Figure 11: Symbols used by BUBBLE

This subroutine draws a colored version of the symbol in Figure 11 (see Figure 12).

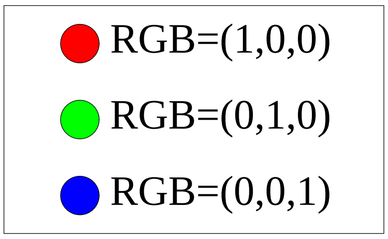

Figure 12: Colored discs using RGB_BUBBLE

- BUBBLES(XDES,YDES,N1,N2,NDIF,
  AMMBIG,AMMSML,GREY,SYM)

  This subroutine takes as arguments two real vectors *XDES,YDES*, three integers *N1,N2,NDIF*, three real numbers *AMMBIG,AMMSML,GREY* and one string *SYM* (see the subroutine BUBBLE).

  It draws *N2* bubbles *SYM* of two different sizes (*AMMBIG* and *AMMSML*) at place *XDES(I),YDES(I)* for *I=1,N2*. More precisely, it draws a bubble of size *AMMBIG* at position *XDES(N1),YDES(N1)* and at every *NDIF* positions in a cycle (see Figure 13).

- BUBBLES_OPEN(XDES,YDES,N1,N2,NDIF,
  AMMBIG,AMMSML,GREY,SYM)

  Same as the previous subroutine but without control of the drawing box.

Exercises

- *Draw a picture with five of the symbols appearing in Figure 11. After this, color your symbols.*

- *Like when we were kids ... Draw a fish which expires bubbles of different sizes (use the subroutine* BUBBLES*).*

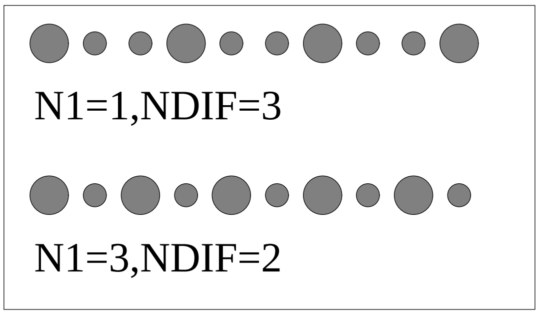

Figure 13: Subroutine BUBBLES used with *N2*= 10

4.6 Axes

- AXEX(X0,Y0,X1,X2,XD,K) and AXEY(X0,Y0,Y1,Y2,YD,K)

  This two subroutines take as argument five real numbers *X0,Y0,X1,X2,XD*, resp. *Y1,Y2,YD*, and an integer *K*.

  They draw the x-axes, resp. the y-axes, where *(X0,Y0)* is the position of the origin, *X1,X2*, resp. *Y1,Y2*, are the delimiters. *XD*, resp. *YD*, is the distance of the large graduations, and *K* is the number of sub-graduations.

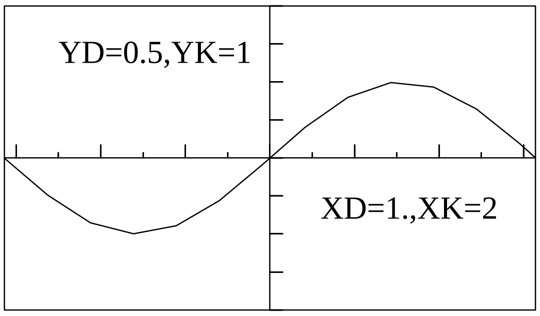

Figure 14: The sine function and the axes using AXEX and AXEY

- LOGAX_X(X0,Y0,X1,X2,KDEL,KGRAD) and LOGAX_Y(X0,Y0,Y1,Y2,KDEL,KGRAD)

  These subroutines take as arguments four real numbers *X0,Y0,X1,X2*, resp. *Y1,Y2*, and two integers *KDEL,KGRAD*. The possible values for *KDEL* (the power of 10) are $1, 2, 3, 4, \ldots$. Those for *KGRAD* are $-1, 0, 1$.

  These subroutines are useful for drawing logarithmical axes with labels.

- LABEL_X(X0,Y0,X1,X2,XD,IDIGIT) and LABEL_Y(X0,Y0,Y1,Y2,YD,IDIGIT)

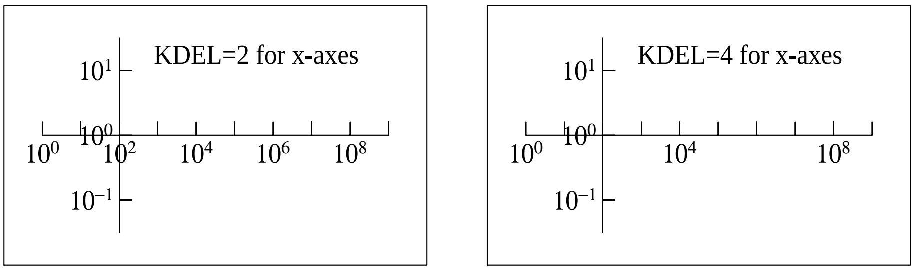

Figure 15: Logarithmical axes using LOGAX_X and LOGAX_Y (with KGRAD=0)

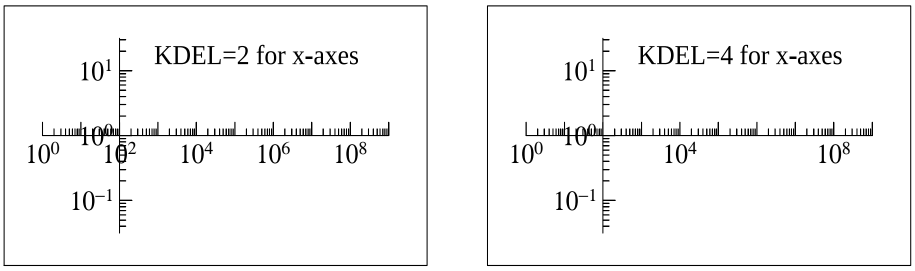

Figure 16: Logarithmical axes using LOGAX_X and LOGAX_Y (with KGRAD=+1)

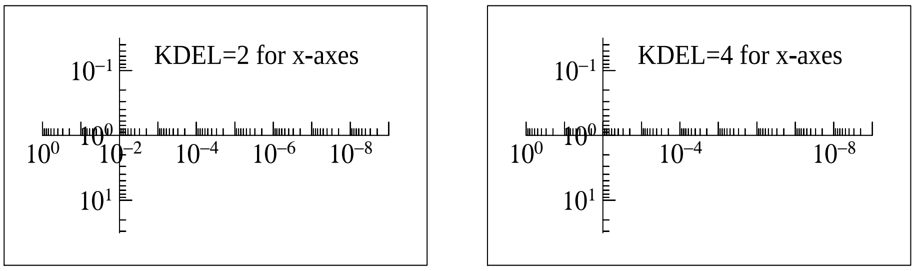

Figure 17: Logarithmical axes using LOGAX_X and LOGAX_Y (with KGRAD=-1)

They take as arguments five real numbers *X0, Y0, X1, X2, XD*, resp. *X0, Y0, Y1, Y2, YD*, and an integer *IDIGIT*.

These subroutines write labels. *(X0,Y0)* is the position of the origin, *X1,X2* are the delimiters, *XD* is the distance of the large graduations and *IDIGIT* is the number of digits (similar for the y-axes).

- LABEL_X_W(X0,Y0,X1,X2,XD,IDIGIT) and
  LABEL_Y_W(X0,Y0,Y1,Y2,YD,IDIGIT)

  Same as the previous subroutines LABEL_X and LABEL_Y, but with letters in white boxes.

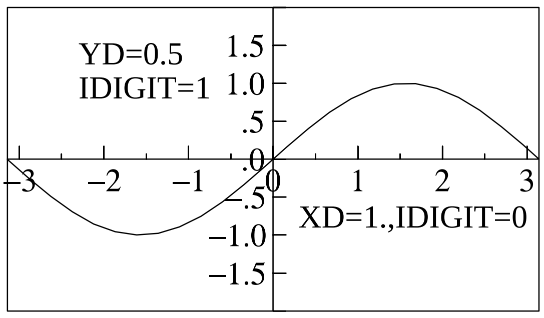

Figure 18: The sine function, the axes and the labels

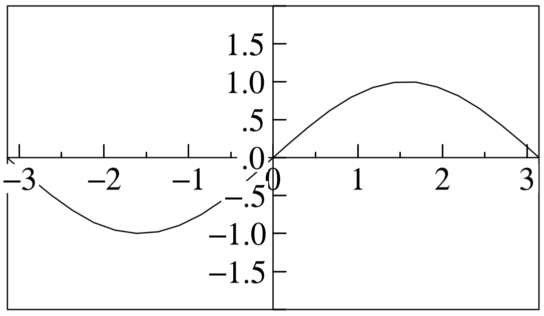

Figure 19: Same example as before, but with LABEL_X_W and LABEL_Y_W

Exercises

- *In the same picture, using* MICKEY_MOUSE*, draw four different axes with their labels (vary the distance between the graduations).*
- *Draw the cosine function in the interval* $[-3, 3]$*, draw the axes and label them.*

4.7 Text

- TEXT(X,Y,CHAR) and TEXT_RELAD(X,Y,CHAR)

  This two subroutines take as argument two real numbers *X, Y* and a string *CHAR*.

  The subroutine TEXT writes the string *CHAR* starting at position *(X, Y)* in ps-fonts. There are special characters: *$r* for roman font, *$i* for italic font, *$s* for symbolic font, ˆ for exponent printing, _ for subscript printing, and ~ if you want to go back to normal font style.

  The position for the subroutine TEXT_RELAD are the coordinates (between 0 and 1) relative to the box.

- TEXT_W(X,Y,CHAR) and TEXT_WW(X,Y,CHAR)

  Same arguments as the previous one: two real numbers *X, Y* and a string *CHAR*.

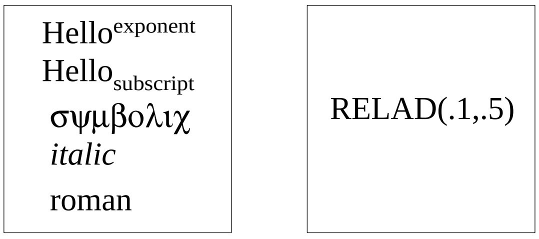

Figure 20: Examples for TEXT and TEXT_RELAD

Similar to TEXT, these subroutines write the string *CHAR* in a white box.

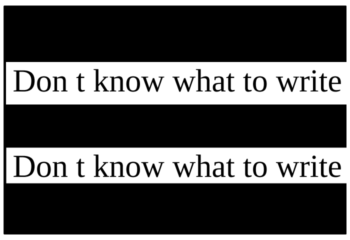

Figure 21: TEXT_W (down) and TEXT_WW (up)

- TEXT_LATEX(X,Y,ALAM,AMU,CHAR) and TEXT_LATEX_W(X,Y,ALAM,AMU,CHAR)

  These two subroutines take as argument four real numbers *X,Y,ALAM,AMU* and a string *CHAR*.

  The first subroutine write the LaTeX formula *CHAR* at position *(X,Y)*. The parameters *ALAM,AMU* are the relative coordinates (between 0 and 1) of the position point *(X,Y)* inside the box and allow all types of centering. Values outside the interval $[0,1]$ are also possible. Many compilers require two '\\' to produce one '\'.

  The subroutine TEXT_LATEX_W puts the formula inside a white box.

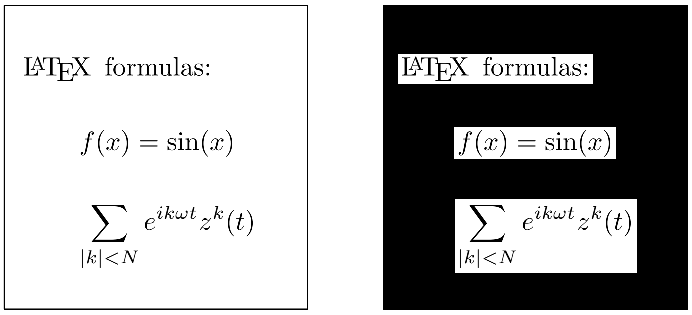
LaTeX formulas:
$f(x) = \sin(x)$
$\sum_{|k|<N} e^{ik\omega t} z^k(t)$
LaTeX formulas:
$f(x) = \sin(x)$
$\sum_{|k|<N} e^{ik\omega t} z^k(t)$

Figure 22: TEXT_LATEX (left) and TEXT_LATEX_W (right)

- TEXT_LATEX_R(X,Y,ALAM,AMU,ROT,CHAR),
  TEXT_LATEX_RG(X,Y,ALAM,AMU,ROT,GRAY,CHAR) and
  TEXT_LATEX_RRGB(X,Y,ALAM,AMU,ROT
  ,RED,GREEN,BLUE,CHAR)

  These subroutines take as argument five real numbers *X,Y,ALAM,AMU,ROT* and a string *CHAR*. In addition, the second subroutine takes a real number (between zero and one) *GRAY* and the third takes three real numbers (also between zero and one) *RED,GREEN,BLUE*.

  These subroutines write the LaTeX formula *CHAR* at position *(X,Y)*. The parameters *ALAM,AMU* are the relative coordinates (between 0 and 1) of the position point *(X,Y)* inside the box and allow all types of centering. It leaned the text by an angle *ROT* (in degree). The second and third subroutine, in addition, draw a gray, resp. rgb-colored, box around the text.

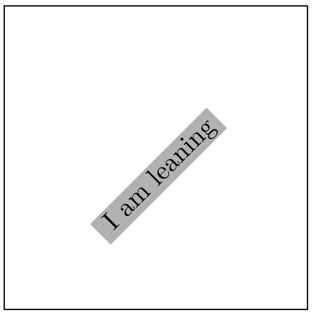

Figure 23: A leaned formula (*ROT*= 45) using TEXT_LATEX_RG

- TEXT_RELAD_LATEX(X,Y,ALAM,AMU,CHAR),
  TEXT_RELAD_LATEX_W(X,Y,ALAM,AMU,CHAR) and
  THICK_BORDER(THICKNESS)

  The first two subroutines are similar to the previous ones but *X,Y* (between 0 and 1) are coordinates relative to the box.

  The subroutine THICK_BORDER takes as argument a real number *THICKNESS* and allows to increase the thickness of the white box (normally 1 point).

- BEGIN_TEXT_LATEX(X,Y,ALAM,AMU,CHAR),
  BEGIN_TEXT_LATEX_W(X,Y,ALAM,AMU,CHAR),
  BEGIN_TEXT_RELAD_LATEX(X,Y,ALAM,AMU,CHAR),
  BEGIN_TEXT_RELAD_LATEX_W(X,Y,ALAM,AMU,CHAR),
  CONTINUE_TEXT_LATEX(CHAR), and
  END_TEXT_LATEX(CHAR)

  These subroutines are useful for LaTeX texts and formulas with more than 80 characters. Use one of the following subroutines

  BEGIN_TEXT_LATEX,

BEGIN_TEXT_LATEX_W,

BEGIN_TEXT_RELAD_LATEX,

BEGIN_TEXT_RELAD_LATEX_W

for the first line (same signification as the subroutines above), then use
CONTINUE_TEXT_LATEX(CHAR)
for continuation lines, and finally
END_TEXT_LATEX(CHAR)
for the last line of the formula.

- TEXT28(X,Y)

  It takes as argument two real numbers *X, Y*.

  This subroutine reads text from file 28 and writes it at position *X, Y*.

  Enables to write texts and numbers created by the program, for example results of a computation. First, write them into file 28 (do not open a file labeled 28 !) in a format required by TEXT or CHAR, then call TEXT28.

```
          ...
C compute the value of HMIN
          ...
C writes the value of HMIN in the file 28
   WRITE(28,*)HMIN
          ...
C call of the subroutine TEXT28
   CALL TEXT28(0.1,3.)
```

- TEXT28_RELAD(X,Y), TEXT28_APPEND, TEXT28_W(X,Y), TEXT28_RELAD_W(X,Y), TEXT28_APPEND_W and TEXT_LATEX28(X,Y,GAM1,GAM2)

  Similar as above.

Exercises

- *Write the famous "Hello world" using the roman and italic fonts.*
- *Write your favorite LaTeX formula using the subroutine* TEXT_LATEX.

4.8 Miscs

- ACCOLADE_X(YOFF,YFERM,X1,X2) and ACCOLADE_Y(XOFF,XFERM,Y1,Y2)

  These two subroutines take as argument four real numbers *YOFF, YFERM, X1, X2*, resp. *YOFF, YFERM, X1, X2*.

  The subroutine ACCOLADE_X draws an horizontal accolade at position *(X1,X2)*. The two real numbers *YOFF* and *YFERM* define the height of the accolade. Similar for ACCOLADE_Y, but draws a vertical accolade.

- MAGNIFICATION(ISCALE) This subroutine reduces the picture by 1.2^{iscale} to obtain correct size after LaTeX's `\magnification` command.

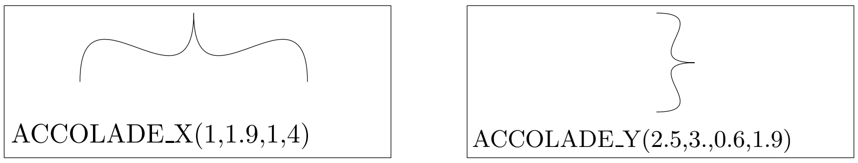

Figure 24: Two BIG braces

- CLOSE_FILL(XDES,YDES,NDES,GREY,STROKE) and
  RGB_CLOSE_FILL(XDES,YDES,NDES,
  RED,GREEN,BLUE,STROKE)

  The first subroutine takes as argument two real array *XDES, YDES* of size *NDES*, a real number *GREY* and a logical argument *STROKE*. The second one, instead of *GRAY*, takes three real numbers (between 0 and 1) *RED, GREEN, BLUE*.

  CLOSE_FILL fills the surface delimited by *XDES(I),YDES(I)*, for *I=1,NDES* with grey tone. *GREY* is the brightness (0 for black, 1 for white). If the logical *.TRUE.* is replaced by *.FALSE.*, there is no boundary line drawn.

  RGB_CLOSE_FILL does the same in color.

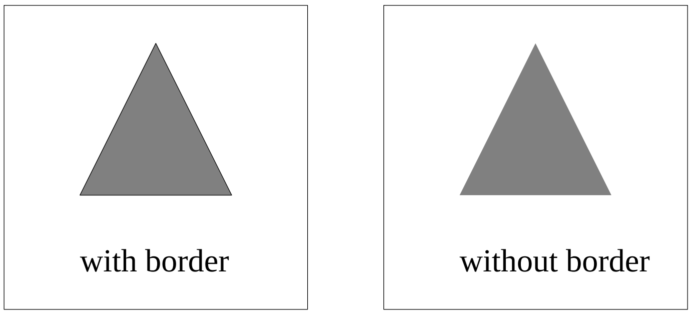

Figure 25: Filling of two triangles using CLOSE_FILL

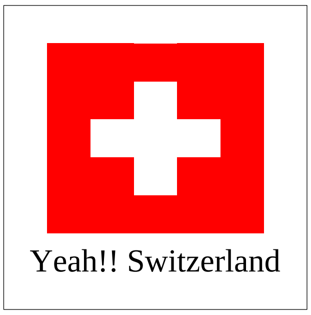

Figure 26: The Swiss flag using RGB_CLOSE_FILE

- SETGRAY(GRAY)

Puts overall plotting gray tone, for example to draw ”white” lines above black ones.

- COLOR(COL) and RGB_COLOR(RED,GREEN,BLUE)

  The first subroutine takes a string *COL* as argument (see below), the second one takes three real numbers (between 0 and 1) *RED,GREEN* and *BLUE*.

  These subroutines set plotting color. The choices for *COL* are: ’blue’, ’green’, ’red’, ’pink’, ’violet’, ’yellow’, ’orange’ and ’black’. For RGB_COLOR, *RED,GREEN,BLUE* are values indicating brightness.

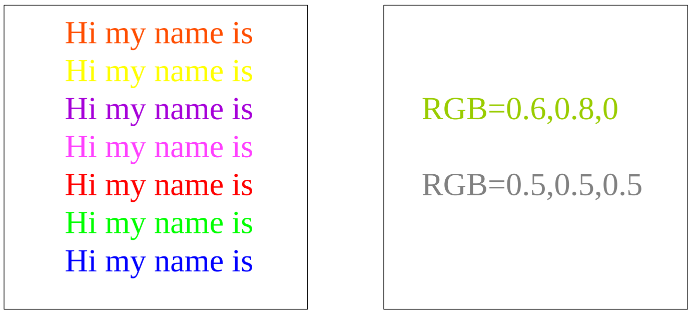

Figure 27: Colored pictures: using COLOR (left) and RGB_COLOR (right)

- COLOR_BOUNDINGBOX(COL)

  It takes as argument a string *COL* which can be *white, black, blue, green, red, pink, violet, yellow* or *orange*.

  As the name of this subroutine indicates it, it permits to draw a colored bounding box.

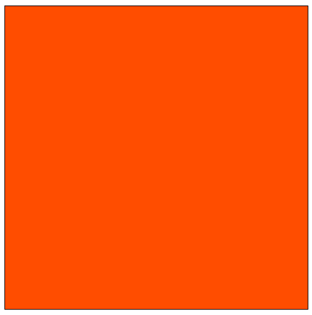

Figure 28: An orange bounding box

- BITMAP_PREPARE, BITMAP_POINT(X,Y,ITHICK), BITMAP_POINT_CLR(X,Y,ITHICK), BITMAP_PIX(IPX,IPY), BITMAP_PIX_CLR(IPX,IPY), and BITMAP_SEND

  BITMAP_PREPARE cleans the memory, BITMAP_PIX(IPX,IPY) adds a singular dot at pixel-number *IPX,IPY*, BITMAP_PIX_CLR(IPX,IPY) clears the same dot, BITMAP_POINT(X,Y,ITHICK) adds a spot of thickness *ITHICK* at position *X,Y*, BITMAP_POINT_CLR clears the spot, BITMAP_SEND sends then the whole picture to the printer.

These subroutine are very useful for drawing millions of dots.

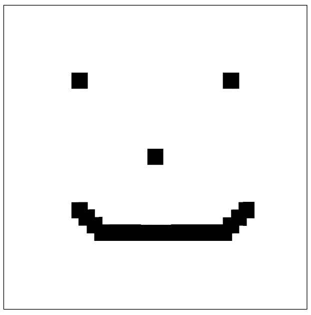

Figure 29: A small example with the subroutine BITMAP_POINT

We conclude this subsection by giving a last example of the use of the BITMAP routines. We compute the GCD of two integers (ranging from one to hundred), and then draw a pixel if this GCD is one:

```
...
CCC we prepare the BITMAP subroutines
      CALL BITMAP_PREPARE
CCC compute the GCD of two integers I and J, for I,J=1,100
      DO I=1,100
        DO J=1,100
          CALL GCD(I,J,PGCD)
  IF (pgcd .eq. 1) THEN
            CALL BITMAP_POINT(1.*I,1.*J,8)
  END IF
        END DO
      END DO
CCC we send the picture
CALL BITMAP_SEND
...
```

We obtain the following Figure

Exercises

- *Write a small text with various color.*
- *Draw the flag of your favorite country.*

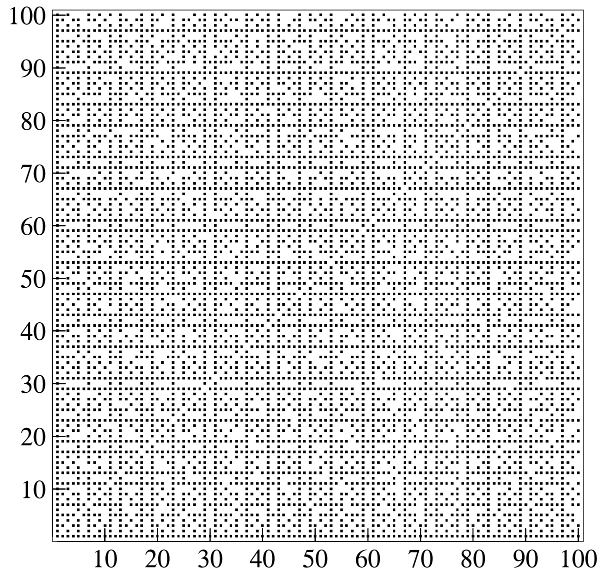

Figure 30: Another example with the subroutine BITMAP_POINT

5 Close the environment

To close the environment, call the following two subroutines
GREATER_BOUNDINGBOX(LEFT,RIGHT,DOWN,UP) and END_GGG.
The first subroutine takes as argument four real numbers *Left,Right,Down,Up* and is used for fine adjustment of the picture inside the text of a LaTeX document. The second subroutine writes the necessary macros and produces the final files.

6 Some useful tips

We finally want to give some useful tips to help using the GGG-routines. If you have additional tricks feel free to contact the authors.

- It may be useful for the compilation, under Linux or MacOSX, to write two small scripts *cfggg* and *plggg*:

```
#cfggg
f95 -o $1.out ~/bin/GGGraphics/GGG.o $1.f
./$1.out
cat $1.vor $1.add > $1.ps
rm $1.vor
rm $1.add
plggg $1
```

  and

```
# plggg
cp $1.ps ~/bin
cp $1.inp ~/bin
cd ~/bin
mv $1.inp fgg.inp
latex figur
dvips -D600 -o figur.ps figur.dvi
gv figur.ps
rm figur.dvi
rm figur.ps
rm fgg.inp
rm *.aux
rm *.log
rm $1.ps
cd -
```

where the LaTeX file figure.tex is just

```
\documentclass[12pt]{report}
\usepackage{epsfig}
\usepackage{~/bin/GGGraphics/GGGraphics}
\pagestyle{empty}
\begin{document}
%***********************************
\begin{figure}[h]
\centering
\GGGinput{}{fgg}
\end{figure}
%***********************************
\end{document}
```

Index