VKPS

3.0 (08.00)

by Victor Kohn

Russian Research Centre
"Kurchatov Institute",
Moscow, Russia
http://www.chat.ru/~kohnvict

POSTSCRIPT *based program for creating the publication quality* **DOCUMENTS** *including scientific* **GRAPHICS**. *The result is achieved by means of simple programming from* **ZERO** *level (direct postscript) to* **HIGH** *level of macroprocedures with parameters.*

Introduction to VKPS and PostScript

Victor G. Kohn

Russian Research Centre "Kurchatov Institute", 123182, Moscow

DOCUMENT: A tutorial on the program VKPS which allows one to present scientific results like figures inside the box of axes or topographs or a publication quality text with figures as a postscript document and an introduction to the PostScript language

VERSION: 3.0 (August 2000)

AUTHOR: Victor Kohn

E-MAIL: kohn@kurm.polyn.kiae.su

URL: http://www.chat.ru/~kohnvict

CONTENTS:

1. Introduction

The program VKPS allows one to prepare quickly the results of scientific calculation or any combination of graphical objects as a multipage document written in PostScript language (PSL). The following treatment of the document may be done by standard technique like printing on the postscript printers or looking by the program GhostView.

As known PSL has been elaborated for a presentation of all printed information like pictures, texts, figures, tables, formulas and so on. It is just the input stream of any network printer. PSL is able to describe the whole book with a lot of pages. On the other hand, it is universal and easy to read and to transfer by e-mail as a usual ASCII file.

In principle, PSL is one of the many programming languages. It is possible to write the "xxx.ps" document directly by means of any simple text editor. However, it is inconvenient in many cases, especially when a picture is the scientific figure which shows a lot of values of some function of two variables. The inconvenience is due to the fact that PSL is a low-level programming language. To describe the figure on PSL one needs to determine all details of the figure because each step of postscript interpretator makes only a small work while the number of primitive instructions may be as large as thousands for a complex figure in the case when the calculated lines are formed by thousands points.

The second problem of such a way is a necessity to work with a very large file containing a lot of figures and texts. When some new fragment is wrong it is difficult to search the error and to verify the file many times.

That is why, instead of direct programming the pictures on PSL some users prefer to have the graphical program which allows to work inside the interactive (WYSIWYG) environment when the user may see the result of his actions immediately in a graphical form. This method has advantages in the case of drawed pictures of random structure. However, it is limited in the case of regular pictures

of computed graphics or rather complicated figures contisting of many fragments.

On the other hand, some simple graphical objects like lines, frames, arcs, simple text strings may be described in a simple and direct way. That is why some additional high-level languages were created which are simpler for the beginners and are supported also by the instrumentality to show quickly the picture described. At the final stage when the text of program becomes correct, the original description is translated from the high-level language to Postscript. As an example, of this approach the package GLE (Graphics Line Editor) may be pointed out. This package is free in Internet. Another example is the program METAPOST which is a part of the package MiKTeX. However, this way is also limited because the user which has learned the high-level language has no motive to use the all possibilities of PSL. In addition, the package GLE, for example, produces the Postscript document (file) of much larger size compared to the size which may be obtained by specialist in the case of direct programming on PSL the scientific figures containing a lot of points. This becomes inconvenient when it is necessary to transfer the file by e-mail or FTP.

I think that the high-level language has to be based directly on PSL in the places where it is convenient. From this point of view it has to contain the possibility to include any piece of ready postscript text in the file of high-level program. Since the Postscript on the whole is very complicated it is difficult to create an independent interpretator of such a language which may work faster and better that GhostView. Therefore the procedure of simple debugging is not possible and it is of interest to realize only the function of translating the high-level language to PSL. In other words, this high-level language must be a set of macro procedures.

These procedures, in principle, may be written directly in PSL where it is desirable together with the compiled procedures. In any case the task is to transform some simple and short text to PSL text which must be also as short as possible. The short text of high-level language allows to avoid errors and save the work of users. The short text of resulting postscript file is convenient for a trasfer by FTP (Internet) to colleagues from other countries.

Such a way may be beneficial for men who work on computer and know the programming languages to create his own calculating programs and to obtain by means of this programs very fresh results which must be considered from different sides. This work demands a lot of graphics of different kinds. Each step deals with one figure but these figures must be accumulated for a further analysis. Sometimes it is necessary to prepare a lecture or a poster to the conference. Another application is to prepare the figures for scientific papers. As it is known, a text of scientific paper may be prepared by means of LaTeX language with more convenience. However, it is not a case for figures. On the other hand, something similar to LaTeX is also accumulated in the new language.

This way supposes the knowledge of the Postscript language at least partially. Nevertheless all routine work is performed automatically by computer. I see two great problems. The first one is to make the total big Postcript file of many pages from the ready fragments. The second one is to describe the separate fragments which may have different nature. The language of the program VKPS just solves both these problems. It involves the direct postscript operators only for describing the simple fragments therefore it is not necessary to know the language on a whole. It is enough to know only the structure of some simple operators. The initial information about the PSL is also included in this document as an appendix.

2. Structure of VKPS language

2.1 General features

The version 3.0 of VKPS language (VKPSL) consists of macro procedures of scientific graphics as well as formatted text which are translated in PSL as a publication quality text with figures. The interaction between the text and figures is simpler compared to LaTeX. VKPSL gives also some additional advantages because the style can be changed from one paragraph to another. On the other hand, the simple graphical EPS fragments for a usage with LaTeX may be quickly prepared as a particular case of the program possibilities. Any manipulations with colors are possible.

The language is based on the macro procedures of two different kind. The macros of first kind are pure postscript procedures (sometimes simply short definitions of postscript operators) with the aim to make a postscript text as short and clear as possible and to use PSL directly in the places where it is convenient. The macros of second kind are more complicated procedures which describe automatically

the well defined fragments like the scientific figures inside the box of axes or graphics of two-dimentional array. These procedures may have a lot of parameters which have to be defined previously.

The text of program in VKPSL (high-level) must be written in a separate file. The program VKPS takes the name of such a file from the command line and it uses the content of the file as an input stream of information for a creation the output stream as a postscript file. The name of PS file is generally specified by the input file. This way allows one to realize automatic call of the program by pressing the name of input file in the operating system like Windows. The extension of the input file name may be arbitrary but for a convenience it is assumed below (vkp) extension. Hence the program creates a postscript file (xxx.ps) from (xxx.vkp) file which may be shown later by GhostView program and (vkps.log) file which contains an information about possible errors and some diagnostics.

The elements of VKPSL are: **commands, parameters, comments, postscript fragments and datas.** Each command or comment or ps-fragment uses the total line. They are distinguished by first symbol of line, namely,

% for a comment similarly to PSL
but only in the first position;
for a command followed by name of command
and field of arguments;
= for a direct postscript line to insert it
in the document.

The comments are not used in resulting postscript text. These are only for a user convenience to make a clear text.

The direct postscript lines will be completely inserted in a postscript file at the current place excluding the (=) sign, hence the second sign of input line will be the first sign of output line and so on. There are 10 commands:

#title: title of the work;
#input filename to input contents;
#save filename to save the following text;
#xaxis XBE XEN XFM XSM XNS;
#faxis FBE FEN FFM FSM FNS;
#yaxis YFF YSF YFM YSM;
#function XFP XLP NXP NFU IFF;
#draw figure;
#new page;
#end

The parameters are divided on the text parameters and the numerical parameters. All parameters are previously defined and user may redefine some of them many times. The name of the parameters as well as the commands may be as long as convenient (without a blank symbol) but only three first letters are essential for the program. For example, the shortened names of commands are: #tit, #inp, #sav, #xax, #fax, #yax, #fun, #dra, #new, #end. Therefore below the shortened names of the parameters will be described together with a short definitions in round brackets and an initial value in square brackets.

The text parameter must be only one in a line. The definition has a usual form (name =value) and blank symbols before and after the name are ignored while the value is the text which begins just after the (=) sign up to end of the line. There are 7 text parameters:

psf (postscript file name) [0001.ps];
hfi (head file name) [head30.psf];
cfi (color file name) [col30vk.psf];
dfi (data file name) [vksign.dat];
xti (x-axis title) [(axis X)Lc];
fti (f-axis title) [(axis F)Lc];
yti (y-axis title) [(axis Y)Lc];

There are 54 numerical parameters. Their definitions have the same form (name = value) but the difference is that there may be many definitions in the line separated by one or more blank signs. The total table of the numerical parameters is given in Appendix. The numerical parameters give the program information on how to draw the figure fragment like the kind of graphical object, the position on the page, the size, the parameters of axes, the curves and so on. The sense of some parameters is different for different graphical objects. It is convenient to describe them step by step later on examples.

2.2 Simple examples

It is of interest to demonstrate how much simple VKPS programs are looking. The most simple correct program contains only one operator

```
#end
```

This program creates the postcript file "0001.ps" which shows nothing. The program containing two operators

```
#draw
#end
```

creates the postscript file "0001.ps" which shows the

figure if the data file "vksign.dat" exists on the current directory. In the opposite case the error will be detected. The figure is defined completely by the initial values of the parameters. Typical text of program looks like this

```
#title: example of VKPS program
  psfile=figures.ps
  hfile=c:\head30.psf
= 70 770 m is3(VKPS example, page 1)L ds3
  dfile=vksign.dat
  kind=1 xco=100 yco=250 xas=350 fas=300
#xaxis 0 105 0 20 9
#faxis 0 75 0 10 4
  xtitle=(X-axis title)Lc
  ftitle=(F-axis title)Lc
  txx=0 txy=0 tfx=0 tfy=0
#draw
#new page
= 70 770 m is3(VKPS example, page 2)L ds3
  dfile=rtxt.rt
  kind=6 xco=70 yco=700 sca=0.7
  lhe=20 lwi=450 mod=0
#draw
#end
```

One may see rather complicated figures (on two pages) may be described by only few instructions. Thanks to an absence of extra information the probability to make an error tends to a minimum.

2.3 Analysis of the typical example

Let us analyze what information is contained in the last VKPS program. The command (**#title**) contains the text after the sign (:) which will be inserted in the comment line of output PSL file
%%Title: example of VKPS program
This is not necessary but allows one to distinguish different PSL files.

The text parameter (**psfile**) defines the name of output PSL file as "figures.ps". The text parameter (**hfile**) defines the name of the head file. The head files contain the postscript definitions which are used by VKPS program and by user in the direct PSL fragments. The user may define up to 10 different head files. However, the file "head30.psf" is necessary because it is used by VKPS-3.0 program. Normally it is distributed with the program itself and must exist at the current directory. However, there is a possibility to place the file in any directory and to specify the total path of the file (up to 64 signs).

The direct PSL line defines the text by means of the definitions of the head file. The font, the position, the size and the symbol line are described extremely shortly. It is convenient to use PSL directly for such a simple graphical objects. The commands of the head files are described in Appendix.

The text parameter (**dfile**) defines the name of the data file as "vksign.dat". The data file must be consistent with the kind of the graphical object. The kind is defined by the numerical parameter (**kind**). When kind = 1 the data file must contain the curve as two-column ASCII file - each dot in a separate line (x,y). The data files must be prepared by user as a result of calculation. However, there is a possibility to determine the data file directly in the VKPS program. For this purpose the following scheme may be used

```
  dfile=here
first line of the data file
second line of the data file
and so on
%eof
```

Here the comment line **%eof** plays a role of the end of file pointer.

The numerical parameters (**xco, yco**) define the position of the figure on the page (x and y coordinates). VKPS program works with the standard postscript units of length, namely, **pt = 0.0353 cm**. The numerical parameters (**xas, fas**) define the sizes of axes for an argument and a function (x axis size and f axis size). The command (**#xaxis**) define directly five numerical parameters in a shortened form (**xbe xen xfm xsm xns**). The field of argument contains five numbers which will be transfered to the parameters. The sense of the parameters are: xbe = (x-axis begin), xen = (x-axis end), xfm = (x-axis first long mark with figures), xsm = (x-axis step to the next long mark with figures), xns = (x-axis number of short marks without figures between the long marks). The units may be any and these have to match the mathematical units of the calculated dependence. The command (**#faxis**) define five numerical parameters (**fbe fen ffm fsm fns**). The parameters are similar to the x-axis but these define f-axis - the vertical axis of the function. The sense of the parameters are completely the same as above.

The text parameter (**xtitle**) defines the title of the x-axis as a text string. The argument is a direct PSL text using the definitions of the head files. In this way the title may contain Latin and Greek and

Russian symbols as well as subscript and superscript and mathematical symbols and so on. There is no restrictions at all. To determine the complicated text string with a usage of many fonts and text commands the number of symbols may be large enough. However, the program allows one to use no more 100 signs in the total line icluding the name of parameter. Usually it is enough. When it is not a case there is another possibility to put the title as long as possible (see below).

The text parameter (**ftitle**) defines the title of the f-axis in the same manner. The simplest definition used in the example is the standard Latin text placed in the round brackets and followed by the PSL command **Lc**. This command defines the Palatino-Roman font of 14 pt size. The text will be placed to coincide the middle point of the text string with the middle point of the axis (centered). This is standard positions of the title.

The standard positions may be changed by the numerical parameters (**txx, txy**) for the x axis and (**tfx, tfy**) for the f axis. Their values define the shifts of the title from the standard positions. The names of the parameters are the abbreviation of the following: txx = (text of x-axis x-shift), txy = (text of x-axis y-shift), tfx = (text of f-axis x-shift), tfy = (text of f-axis y-shift). The values must be determine in the standard units of length (pt).

The command (**#draw**) declares to draw the graphical object. The current values of all parameters are used for the figure drawing. The reader may understand that this command is position sensitive whereas the definitions of many parameters and some commands are position insensitive. The order of the definitions may be arbitrary with only one restriction - the all desirable definitions must precede the (#draw) command.

The command (**#new page**) declares to begin the new page of the document. It is also position sensitive. Normally at least one command (#draw) must precede the command (#new). In the opposite case the first page of the document will be empty. Usually it is convenient but not necessary to place the (#new) command just after the (#draw) command.

The second page of the document will contain the graphical object of the kind = 6. This is the formatted text. The data file in this case contains the text including the set of commands for the text formation like the fonts, size, the order of the placing in the columns and so on. This is decribed below in more detail. The new numerical parameter (**sca**) allows to scale the all or part of defined position or size parameters. The numerical parameters (**lhe, lwi**) determine the height and width of the text line. The numerical parameter (**mod**) determine the modification of the text formatting.

The command (**#end**) declares that the program is finished. At this point VKPS program starts to prepare the PSL document. The rest part of the input file will be ignored.

2.4 Manipulation with the text string

It is always the problem to describe the complicated text string by usual symbols of the ASCII table. The program VKPS uses two different ways to solve this problem. First one is to use the special text environment as the graphical object of kind = 6 (see below). The second way is to use PSL directly to describe the text string. However, the initial PSL commands are long and not convenient. This is why the head file (head30.psf) contains some postscript definitions to simplify a description of the text. First of all the standard size of fonts are accepted as 14 pt. Then some simple commands were introduced to define the different fonts. For example, (..text..)**L** means to put the ..text.. by Palatino-Roman font of 14 pt size at the current cursor position by the left side of the text string. Similar command (..text..)**G** will show the symbols of the Symbol font with the same ASCII codes as ..text... For example, **(a b c d)G** will show α β χ δ.

There are the commands to use Helvetica font as (..text..)**H**, Palatino-Bold font as (..text..)**B**, Palatino-Italic font as (..text..)**I**, Russian text by Helvetica type Russian font as (..text..)**R**. However, in the latter case the additional head file (cyr30fnt.psf) containing the Russian font must be defined because this font is not a standard Postscript font. The correspondence between the ASCII codes and russian letters may be obtains by test VKPS program. For example, (a b c d)**R** will show а б ц д.

The command **(abc)ib** allows to draw subscript by current font and smaller size. The command **(abc)it** is for superscript. The current position of the cursor may be redefined by the command X Y **m** or X Y **rm**. The first command define the absolute position of the cursor in pt - X as x-coordinate, Y as y-coordinate. The second command define the shift of the current position on the values X and Y.

One has a possibility to change a standard size of the font. There are 5 commands to increase the size

is1, is2, is3, is4, is5 and 5 commands to decrease the size **ds1, ds2, ds3, ds4, ds5**. The corresponding pair of commands make nothing, for example, is3 ds3. In the case of increasing the size of font it is multiplied by the values 1.25, 1.5625, 2, 3.125 and 4. In the case of decreasing we have correspondingly 0.8, 0.64, 0.5, 0.32 and 0.25. It is necessary to use only the corresponding pair of the commands, for example, is3 (text)L ds3 to eliminate the scaling of the following fragments.

A slightly different command (..text..)**Lc** means to put the text by central point of the text at the current cursor position. It is used to place the title of axes at the centre of axes. This command is only for L font (Palatino-Roman of 14 pt size). However, using the combination ()**B**(text)**sc** the same property may be obtain for all other fonts. The command (..text..)**Le** means to put the text by it's end at the current cursor position with the L font . For other fonts the combination ()**B**(text)**se** is possible.

The direct PSL text may change the color. The color is defined directly as R G B **srgb** command where R, G and B are numbers from 0 to 1 which define the concentration of red, green and blue colors in the real color.

The additional possibility may be obtained with the head file (head-a.psf). This file defines the command **AA** for the Angstroem, **hh** for the Planck constant, **int** for the integral sign, **min1, min2** for the minus sign above the small and capital figures - the notation used in Crystallography and others. The number of nonstandard symbols may be increased and any new symbol may be described in the head file. Hence there is really no restriction.

2.5 Manipulation with the files

There are two commands which don't produce the graphics directly but are convenient to prepare the input file. The command (**#input**) has one argument - the name of the file. It is position sensitive. The command declares that the following information must be read from the file specified in the command instead of the input file (xxx.vkp). The file will be read from the beginning to the end. After that the program continues reading the input file (xxx.vkp). This command allows one to prepare quickly the combined PSL document from the different parts which are prepared previously in the separated files. Especially it is convenient when some part of the input file must be repeated many times. This part may be prepared in a separate file and then it is necessary only to use the (#input) command many times.

The second command (**#save**) just allows user to save some fragment of the input file (xxx.vkp) in the different file. The argument of the command defines the filename of the new file. If the file with such a filename exists before the usage of the command then it will be replaced on the new content automatically. The text of the input file which follows after the (#save) command will be considered as the simple text but not the program. The commands will be only saved but not fulfilled. Similar to the construction (dfile=here) the comment line (%eof) from the first position plays the role of the end of file pointer. The command (#save) is a development of the construction (dfile=here) and it may be used in particular to save the data file as a normal file and then to use such a file. The construction (dfile=here) works similarly but it use the filename 'vkpsd.tmp' which is deleted before the program finish. The command (#save) may create or redefine the contents of the file but it cannot to delete the created file.

The interaction of the commands (#save) and (#input) with the same filename may be used as the procedure or subroutine in other programming languages. The command (#save) is used to define the procedure previously and the command (#input) is use to fulfill the procedure many times.

2.6 Manipulation with the colors

A usage of the colors as well as a manipulation with the files are the new features of the version 3.0. The VKPSL has two new numerical parameters: (**col**) and (**lco**). The first parameter defines the general color which will be used to draw the axes and texts. The second parameter defines the color which will be used for drawing the lines of the curves showing the mathematical dependences. There is also the text parameter (**cfile**) which defines the name of the file containing the table of colors for drawing the color topographs.

To determine the color VKPSL uses the six-digit integer number which has a structure "rrggbb" where "rr" is two-digit integer which defines the degree of red from 0 to 99, "gg" is the same for a definition of green and "bb" is the same for a definition of blue. The real color will be mixed color combined from these three colors. For example, 990000 will define bright red, 660000 - middle red, 330000 - dark red, 9900 - bright green, 6600 - middle green, 3300 - dark

green, 99 - bright blue, 66 - middle blue, 33 - dark blue, 999900 - bright yellow and so on. The number 0 is for black and 999999 is for white. The PSL used RGB representation where each color is defined from 0 to 1. In accordance with this representation VKPS multiplies the two-digit integer number for each color by 0.01. Hence we have 0.99 0.99 0.99 for the white color instead of 1 1 1. However the pure white 1 1 1 does not differ from 0.99 0.99 0.99 for many printers and dispays. At any rate such a difference is very small. One may conclude that the zero sign at the left is not necessary because this is a number not the text.

Such a representation is used in all places of the VKPS program, for example, in the (color file) or in the definition of the regions and lines directly.

3. Kinds of graphical objects

In this section we consider the possible graphical objects which may be described simply for automatic PSL figure creating. The notion of the graphical object is very useful because it match well the fact of existence of different data and different forms of the data presentation. The drawing of the graphical objects is a rather complicated procedure which is made by VKPS automatically. The task of user is to give VKPS only the minimum information about what the user likes to have in the figure. Please don't forget that all parameters are defined from the beginning. The user have to redefine only the parameters which are wrongly defined.

3.1 Direct postscript program

This object has **kind = 0**. The data file which name is defined by the the parameter **dfile=name** has to contain the pure postscript text of the graphical object but without a Prologue of the PS document (see Appendix about the PS file structure). This text may use the definitions and procedures of head files to simplify the description. Only the parameters (**xco, yco, sca**) are necessary which define the translation of the figure on XCO YCO values in pt. Here and below the name of large letters will denote the values of the parameters of the same name (of small letters). The (sca) parameter defines a scaling of all figure on the SCA value in both directions. The command (#draw) in this case simply puts the text of data file inside the whole postscript document with one line before and after, the text, namely,

```
gsave XCO YCO translate SCA SCA scale
  <text of the data file>
grestore
```

The module is useful to include the photo or any other bitmap picture saved as PS image. In this case the data file has to be the PS document of image but without the Prologue. Sometimes this construction becomes beneficial compared to the direct PS lines for the drawing the complex texts like mathematical equation. The procedure of the text formatting of VKPS program cannot described the formula of any structure but the PSL can make this. For example, for the equation

$$\int_0^\infty dt\, e^{i\omega t-\gamma t} = \frac{i}{\omega+i\gamma} \qquad (1)$$

the VKPS program may be as follows

```
kind=0 xco=10 yco=10 sca=1
   dfile=here
40 15 m int ds2 -2 -15 rm (0)L -2 35 rm
(\245)G 3 -20 rm is2 ds1 (dt)L 4 Sh (e)L
10 Sv is1 ds2 (i)L(w)G(t-)L(g)G(t)L is2 ds1
-10 Sv ( = )L /X{10 Sh(i)s}bd /Y{(w)G(+i)L
(g)G}bd 3 frac is1 250 15 m ds1 ((1))Le is1
%eof
#draw
```

The text of data file contains the commands of head files. However with other definitions it may be written in another way. At any rate it's structure is a structure of PS language and it is different from LaTeX structure. Another example of PS picture with text is shown below.

The PS program in this case is sligtly more complicated but it is not long because some new procedures first are defined and then used. The text of these procedures may be received from author.

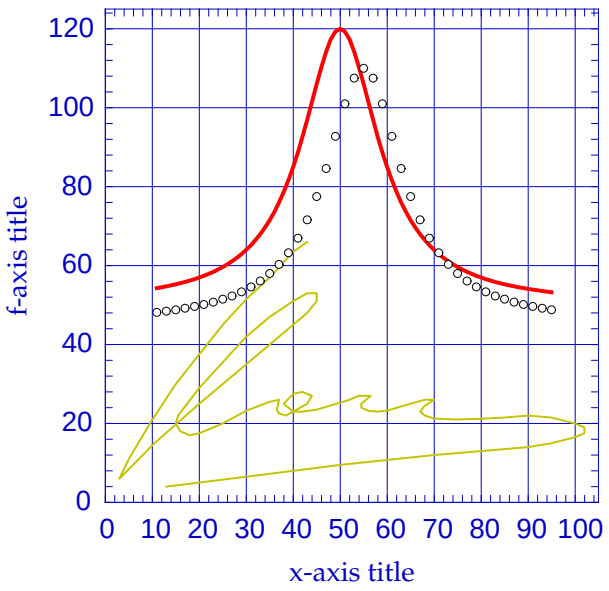
3.2 Graphics in the box from columns

The most popular task of computed graphics is to show a set of functions of one argument inside the box of axes. The function may be theoretical, i.e. continuous, or experimental, i.e. discrete. The box is formed by four axes, each of them must be

independent. The labels may be directed outside of the box, inside the box or form the grid. Usually the bottom and left axes have the titles. Practically each graphics package has the function to create the figure of such a kind with different services. The program VKPS solves this task under the condition of reasonable sufficiency. The program performs the more difficult work to draw the curves, axes and titles. All other comments to figure can be performed by user in another graphical objects.

Such a graphical object has (**kind = 1**). One command (#draw) makes the figure of one function inside the box of axes. The data file must be an ASCII file. It has to contain the two columns, one for an argument and one for a function, each point at separate line. The number of points is defined by the number of lines in the file.

The figure describing such a graphical object is shown below



The fragment of the VKPS program describing such a figure is as follows

```
    dfile=tut1.dat
  kind=1 xco=80 yco=80 sca=1 xas=250 fas=250
  sdi=1 gxs=1 gfs=1 fun=1 lth=1 mas=0
  xbs=1 xts=1 fls=1 frs=1 col=77 lco=777700
  txx=0 txy=0 tfx=0 tfy=0
#xaxis 0 105 0 10 4
#faxis 0 125 0 20 4
  xtitle=(x-axis title)Lc
  ftitle=(f-axis title)Lc
#draw (curve-1)
    dfile=tut2.dat
  xbs=0 xts=0 fls=0 frs=0
  lth=2 mas=0 lco=990000
#draw (curve-2)
    dfile=tut3.dat
  lth=0.5 mas=2 lco=7700
#draw (curve-3)
```

Here all the parameters which are used in this graphical object are shown before the first (#draw) command. Before the second and third commands (#draw) only the parameters having new values are redefined. All the commands of the program presented above were described in the Section 2.3. Let us discuss the parameters which define the properties of the figure. The text parameters and the part of numerical parameters were discussed in the Sections 2.3 and 2.4. Here we describe the new parameters. The parameter (**sdi**) means stroke direction at the marks of the axes. When sdi = 1 the strokes are drawn inside the box of axes. For other values the strokes are outside the box. The initial value is 0. The parameters (**gxs, gfs**) define the existence of grid inside the box (grid of x-axis switch and grid of f-axis switch). The grid lines go through long marks. If gxs = 1 then the grid through x-axis from bottom to top will be drawn. If gfs = 1 then the same will be from the left to the right. For other values the drig is absent. The initial values equal 0.

The parameter (**fun**) means (function unit). It defines the value FUN. The initial values of data file are divided by FUN before positioning at the f-axis. Therefore in reality the program shows the function f(x)/FUN instead of the initial function f(x). It is convenient to match quickly the f-axis parameters and the function values when the calculated values are known previously. There is a special case. If fun = 0 then the decimal logarithm of the values of data-file is drawn instead of initial values and figures at f-axis has the form 10^n where the degrees n are defined values. The initial value of (fun) is 1. However, the intermediate marks don't match the decimal logarithm in the case fun = 0. Therefore the number of intermediate marks must be define as 0. The nonuniform marks of the logarithm scale may be drawn as direct postscript procedure.

The parameter (**lth**) means (line thickness). It defines the thickness of the line in **pt** which are drawn inside the box of axes and shows the function dependence. When lth = 0 the function will be not seen. The initial value is 0.5. The parameter (**mas**) means (marker size). It defines the radius of circle which is drawn at each dot of the dependence by

the line of thickness (lth). If the mas = 0 then the straight line segments will be drawn between each pair of dots and the dependence will be seen as the curve. The initial value is just 0.

The parameters (**xbs, xts, fls, frs**) means (x-axis bottom switch, x-axis top switch, f-axis left switch, f-axis right switch). These parameters declare what axes will be drawn on the figure. The box of axes may be drawn completely or partially or may absent at all. This property is necessary to combine many fragments closely to each other when some axes are common for two neighboring figures. Another case is when several curves are drawn on the same box of axes as it occurs in the program presented above. As it is seen in the program the values 1 means that the axis exists, whereas the value 0 means that the axis is absent. The initial values are 1.

3.3 Graphics in the box directly

The graphical object of (**kind = 2**) is rather similar to the preceding object. The difference is that many functions at the same set of argument values with a constant step may be drawn from the direct access numerical data file. The data file of direct access is the most compressed form of saving the numerical data. It is assumed that the argument values begin from some value and each subsequent value is different from the preceding value on the same value (step). Therefore it is not necessary to save the argument value in the data file. The data file contains only the function values, one function after another so that the values present two-dimensional array - matrix. It is convenient to have a possibility to draw any part of the matrix instead of total matrix, so that the total matrix is a particular case.

To describe the sub-matrix of the whole matrix and the argument values the parameter (dfile) is accompanied by 7 numerical parameters **xfp, xlp, nxp, nfu, iff, nmx, ifx**. The value of parameters determine

XFP - argument value at the first point
XLP - argument value at the last point
NXP - number of argument points to draw
NFU - number of functions to draw
IFF - index of the first function in the total matrix
NMX - argument size of the total matrix
IFX - index of first point in the total matrix

The size of the data file is no less than 4*NMX*(IFF+NFU-1) bytes. It is assumed that the real numbers have REAL*4 format (4 bytes per one number) and these are written in the direct access technique. This means, for example, that the operator (open) of the FORTRAN procedure had the form

```
open(1, file=<data file>, access='direct', recl=nb, ...)
```

where nb = 4*NMX is a number of bytes in the record. First NMX numbers belongs to the first function, second NMX numbers - to the second function and so on. However, only NXP numbers with IFX dot as the first will be drawn. The functions will be shown from the function with the index IFF to the function with the index (IFF+NFU-1). The number NMX is limited by 3960, the number of functions in the file is not limited.

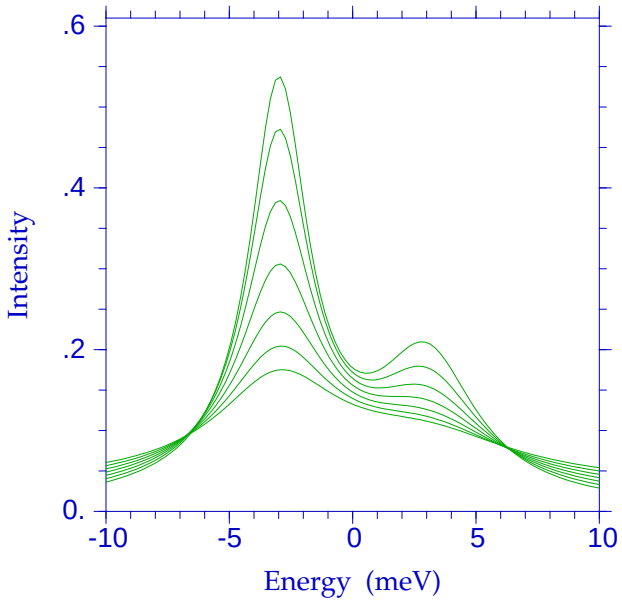
To describe quickly all the parameters discussed above the command

#functions XFP XLP NXP NFU IFF NMX IFX

exists in the VKPSL. As above the fragment of the VKPS program which describe the graphical object is shown below

```
    dfile=tut4.dat
  kind=2 xco=80 yco=80 sca=1 xas=250 fas=250
  fun=1 lth=0.5 mas=0 sdi=0 gxs=0 gfs=0
  xbs=1 xts=1 fls=1 frs=1 col=77 lco=6600
  txx=0 txy=0 tfx=0 tfy=0 vsw=1 vsf=0.005
#xaxis -10 10 -10 5 4
#faxis 0. 0.61 0 0.2 3
#fun -10 10 100 7 34 100 1
  xtitle=(Energy (meV))Lc
  ftitle=(Intensity)Lc
#draw
```

The figure as a result of running this program looks like this



The reader have got an information about all parameters except two, namely, **vsf** and **vsw**. Let us discuss them. Sometimes it is reasonable to present many functions in such a way that each subsequent function becomes shifted vertically on some step compared to preceding function. In this case the function become distributed over the vertical axis and a quasi-illusion of three-dimensional picture arises. Such a vertical shift may be considered as a projection of the y-coordinate (unexisting in reality) on the vertical axis. This property may be obtained by nonziro value of the parameter **vsf** which means (vertical shift of functions). The parameter is defined in mathematical unit of the vertical (function) axis. The initial value is 0 but in the example considered the small vertical shift exists.

To make the illusion of three-dimensional picture more strong the parameter **vsw** allows to open a modification when the values of curve below the values of preceding curve are omitted as invisible. To make this one needs to put 0 value on the parameter vsw (vsw = 0). It is just the initial value. When (vsw = 1) as in the example presented above the lines are visible completely.

3.4 Quasi 3D-graphics of f(x,y)

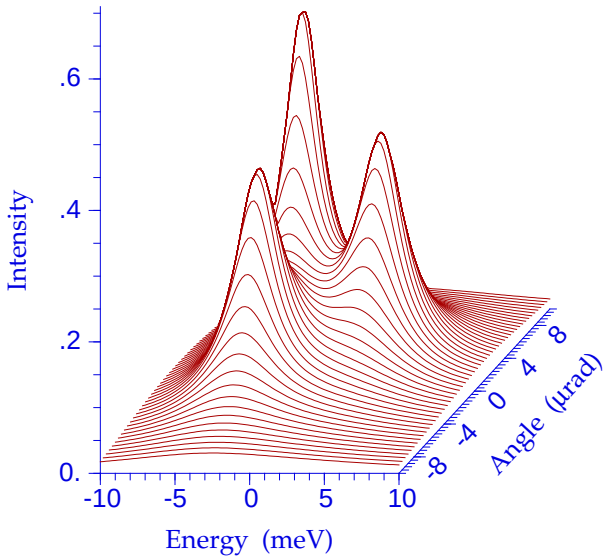
The popular task of computed graphics is to show the function of two arguments f(x,y) as the surface of three-dimensional space. In reality, only the central projection of the surface or the projection in parallel lines along a definite direction can be shown. There are several ways to make this. First one - to deal with the real surface z = f(x,y) in space (x,y,z) and to show its central projection from a definite point of view. It is beneficial for drawing a house or a car with displaying the effects of perspective. The projection with parallel lines corresponds to first way at an infinite distance from the point of view to the object. First way gives two independent parameters - a distance and an angle of view. Second way gives only one parameter - an angle of view.

From the point of view of presentation of scientific results these sorts of graphics is not convenient because it is difficult to compare the difference in dependence at one values of the first argument between the functions which correspond to different values of second argument. That is why VKPS makes the procedure which draws the quasi 3D-graphics as follows. Each curve (each value of second argument) is drawn on the sheet of the same type. Then the sheet for next value of second argument is shifted on the definite vector (sh, sv). Simultaneously the invisible lines can be eliminated and three axes are drawn: X-axis (horisontally), Y-axis (inclined), and F-axis (vertically). All axes have the titles. Such a graphical object has **kind = 3**.

The VKPS solves this task under the condition of reasonable sufficiency. The program performs automatically the more difficult work of drawing the curves, axes and titles. All other can be performed by user using other graphical objects. Once again below the text of VKPS program fragment is shown which describes the figure presented just after the text. All the parameters which influence the module are shown.

```
    dfile=tut4.dat
  kind=3 xco=80 yco=80 sca=1 xas=160 fas=250
  fun=1 hsf=1 vsf=0.005 vsw=0 col=88 lco=660000
  txx=0 txy=0 tfx=0 tfy=0 tyx=0 tyy=0
#xaxis -10 10 -10 5 4
#faxis 0. 0.71 0 0.2 3
#yaxis 6 10 -8 4
#fun -10 10 100 51 1 100 1
  xtitle=(Energy (meV))Lc
  ytitle=(Angle \()Lc(m)G(rad\))L
  ftitle=(Intensity)Lc
#draw
```

The figure itself looks as follows



Similar to the preceding graphical object the data file is the file of direct access. The command (#fun) and the parameters (vsf) and (vsw) has the same sense. However, to make quasi 3D-graphics it is necessary

to perform a horizontal shift together with a vertical shift of the next function compared to the preceding function to have a better view. This is defined by the parameters **hsf**. The horizontal shift is measured in steps of function argument as an integer number contrary to the vertical shift which is measured in units of f-axis. The reason of this difference is related to the peculiarity of the algorithm of eliminating the invisible lines. This restriction makes the algorithm to be much simpler. For a better view one can use the modification when the parts of curves below the preceding curves are omitted as invisible. For this purpose as before there is a parameter **vsw**.

The mechanism of drawing the functions is similar completely to the described above graphical objects of (kind = 1, 2). However the logarithm representation is absent. The axes are drawing in all cases. Therefore the parameteres (xbs, xts, fls, frs) are not used. There are X, F, and Y axes. Because of the figure is only quasi 3D-graphics the sense of inclined y-axis has to be explained separaterly. The axis is drawn through the ends of shifted x-axes for different functions. The number of functions may be arbitrary but not so much owing to the worse view otherwise. That is why the short mark is placed at each function. The user must define the indexes of functions which will get the long marks: the first index by parameter **yff** (y-axis first function) and step to the next index by parameter **ysf** (y-axis step to next function). However what value corresponds to the long mark is unknown previously because the data about the y-axis are absent. Therefore the parameters **yfm** (y-axis first mark) and **ysm** (y-axis step to next mark) determine explicitly these values as first one and step to the next one. These four parameters may be defined by the command

#yaxis YFF YSF YFM YSM

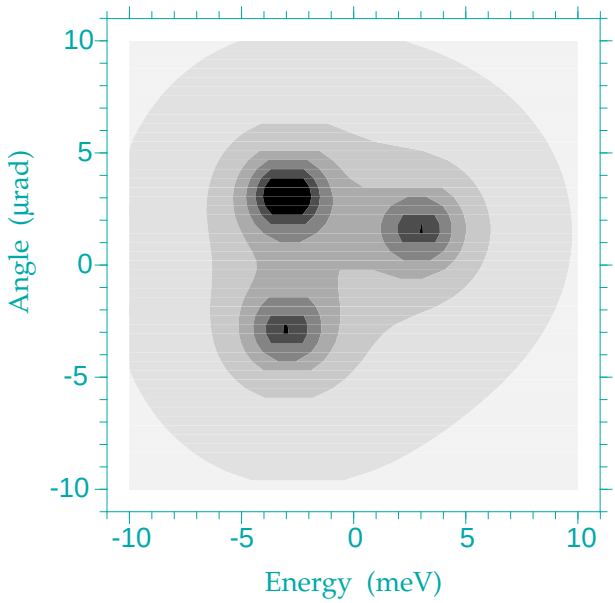
The titles are placed automatically at the good position under the x-axis, before the f-axis and along the the inclined y-axis. However, if one don't like the position of the titles then these positions can be changed. Just the parameters **txx, txy, tfx, tfy** described above for x- and f- axes and new parameters **tyx, tyy** for y-axis are the shifts of the titles position in pt-units. The text of titles must be written as a direct PSL text (see above).

3.5 Topographic image of f(x,y)

The figure presented in the section 3.4 shows well the vertical size of the surface describing the dependence. However, the horisontal positions of the characteristic points of the dependence are shown not explicitly. That is why sometimes it is more beneficial to use another presentation of dependence, namely, the topographic image where the regions of different heights have different gray levels or different colors and the boundaries between the regions of different gray levels show the lines of equal height. Such a dependence just shows well the characteristic points like the positions of maximums and minimums of the function f(x,y). Such a topographic image inside the box of axes can be obtained by using the graphical object of **kind = 4**. Once again below the text of the program is shown together with the figure as a result of the program work. Afterwards some details are explained.

```
    dfile=tut5.dat
    cfile=bw30vk.psf
  kind=4 xco=80 yco=80 sca=1 xas=250 fas=250
  mod=0 xbs=1 xts=1 fls=1 frs=1
  fun=0.4 vsw=7 sdi=0 gxs=0 gfs=0 col=7777
#xaxis -11 11 -10 5 4
#faxis -11 11 -10 5 4
#yaxis -10 10 1 1
#fun -10 10 50 50 1 100 1
  xtitle=(Energy (meV))Lc
  ftitle=(Angle \()Lc(m)G(rad\))L
#draw
```

The figure looks like this



In this case one has to use the same parameters, however some of them have slightly different sense.

The x-axis is the same completely and it describes the x argument of the function. The f-axis (vertical) describes now the y argument of the function. The parameters describing the box of axes are the same (see the description above). The function itself must be in the direct access data file with the same rules of data description by the command (**#fun**). As for the arguments, the x argument is defined as before by parameters **xfp, xlp, nxp**. The y argument is defined by the parameters **yff, ysf, nfu** which means first and last values of the interval and a number of points. As before the data file may be read from the middle and the parameter **iff, nmx, ifx** shows the index of the first lines, the real size of the line and the index of the first point in the line.

To obtain the topographic image it is necessary to determine a set of parameters which define the values of height for the lines of equal height (level parameters). In general this information must be presented in the separate data file which name is defines by the text parameter **cfile**=filename. This file must have two columns - first for the level parameter and second for the color which is used when the function value becomes smaller than the real level parameter. The real level parameter is calculated by multiplying the value from the file by the value FUN. The number of lines in the file must be VSW + 1. The first level parameter will be replaced by -10^{30} as extremely low parameter whereas the last level parameter will be replaced by 10^{30} as extremely high parameter. The first color therefore is not used.

There are two different modifications. When **mod = 0** or **mod = 1** the topograph are drawn as a black-white map when the colors have three components (red, green, blue) with the same concentration which are equivalent to different levels of gray. That is why in these cases the "color" column may contain two-digits integer number as a number of percents of gray level. The figure above was drawn with the file "bw30vk.psf" which was save previously from the "xxx.vkp" file as follows

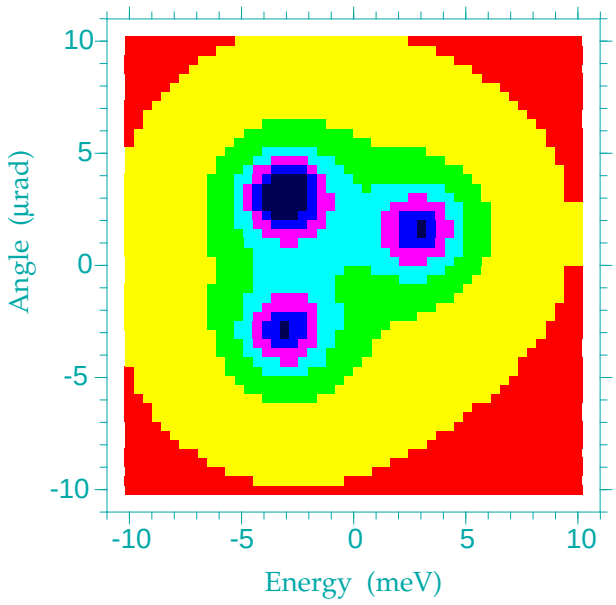
```
#save bw30vk.psf
 -100.        0
   0.08      95
   0.25      88
   0.42      79
   0.58      67
   0.75      52
   0.92      30
  100.        0
%eof
```

There are two algorithms of drawing the topographic image. The main algorithm gives a fine structure of levels lines. This is defined by declaring **mod = 0** or **mod = 2** . The algorithm works well in the case of rather simple surface f(x,y) like the surface shown in the picture presented above. However, this algorithm is not absolutely stable and sometimes it may give errors for the complex function f(x,y) and some set of levels. Unfortunately the algorithm cannot be improved. This is why to solve the problem in the case of rather complex function f(x,y) the more rough but absolutely stable algorithm may be used declaring **mod = 1** or **mod = 3** . To obtain the gray topographs with the stable algorithm the program may be continued like this

```
  mod=1
#draw
```

In this case one uses the same set of parameters as above with only one difference, namely, mod = 1 instead of mod = 0. In the cases with **mod = 0** or **mod = 1** one may obtain only black-white pictures with different gray levels. It is enough for the black-white printers and leads to the shorter size of file.

There is a possibility to draw color pictures like this.



To obtain the color picture one needs to declare **mod = 2** for a fine algorithm and **mod = 3** for a rough algorithm. The colors are described by usual way with six-digits numbers. The picture above was obtained with a next continuation of the program

```
  cfile=col30vk.psf
  mode=3
#draw
```

Here it is assumed that the "xxx.vkp" file defines the color file inside it's body by set of lines

```
#save col30vk.psf
 -100.        0
  0.08   999900
  0.25   990000
  0.42   990099
  0.58   666699
  0.75     9999
  0.92     6699
  100.       66
%eof
```

The color topograph is the new feature of the version 3.0 of the program.

3.6 Regions, lines, text strings, and scaled arcs inside the box

It is easy to describe different regions, lines, text strings, arcs and other simple objects on the page directly by means of postscript language in the real units **pt = 0.0353 cm**. However, sometimes it is necessary to draw the regions or lines or text strings or scaled arcs at the place which is well determined in the mathematical units of the box of axes instead of physical coordinates of these objects. This way is convenient also from the point of view of portability the figures in the box together with the extra details of the figure like regions, lines, texts and arcs. The graphical object of (**kind = 5**) described in this section is just elaborated for this purpose. The box of axes may be drawn and in addition the data file contains a compressed information about how to positioning simple graphical objects relative to these axes.

In the most simple case this object contains only the box of axes if the instruction (**dfile=none**) is introduced. This graphical object may be useful independently as well as in a combination with the other objects using the same or different box of axes. Here the parameters which describe the box of axes coincide completely with the same parameters for (kind = 1, 2, 4). The data file must contain the information about the regions, lines, text strings and arcs in a compressed form - only numbers without comments. Below just the structure of the data file is described.

The data file is a simple ASCII file. The first line must contain four integer numbers in free format:

(nregions), (nlines), (ntexts) and **(narcs)**.

The numbers are the numbers of regions, lines, text strings and arcs which will be drawn in this graphical object. The following lines describe the objects.

If **(nregions)** is larger than 0 then the next line must contain two numbers: (npoints) and (color) which determine the number of points and color for a first region. The color is described in a usual manner (see section 2.6). The next lines must contain x, y coordinates of each point being a vertex of the polygon - boundary of the region. The numbers may be written in free format and each line may contain arbitrary number of data. Then the same structure must be for a second region and so on.

After the all regions are described the similar description must be for all lines if **(nlines)** is larger than 0. The description is the same with one difference - first line contains three parameters (npoins), (line thickness in pt) and (color). The regions and lines cannot go out the box. If some coordinates go out the box then these coordinates are change to the boundary of the box.

If **(ntexts)** is larger than 0 then the following lines must define the text strings. The text strings are defined each string on a separate line. Each line must contain three numbers (x, y coordinates in units of box and color) followed by the text program itself in apostrophes. The text program must include all the postscript commands and may be rather complicated. The change of font or moving the text string is possible. In reality here may appear the whole postscript program for drawing not only the text. The simplest form is like this '(text)s'. The maximum length of text program string on the first line is 82 symbols. However, if the last symbol equals (\) (backslash) then the reading of the text program will be continued on the next line which may contain up to 100 symbols and the last symbol (\) for a continue on the next line and so on. The coordinates of text string may go out the box. Therefore the text string may be used for making the rather complicated title at the axes or above or below the box of axes.

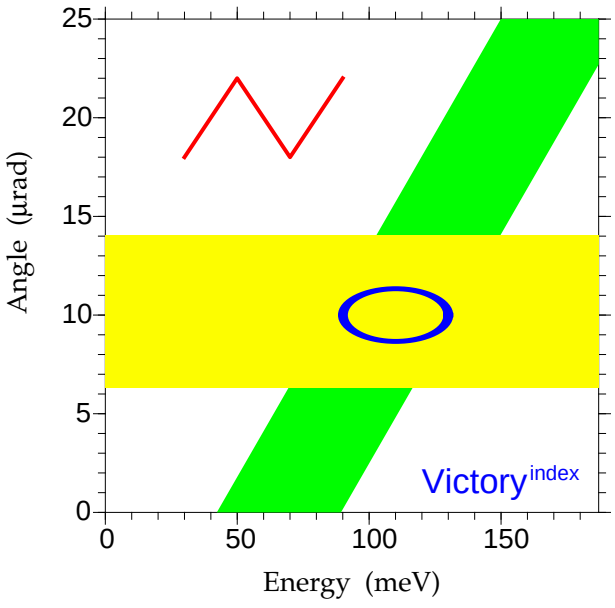
If **(narcs)** is larger than 0 then the following lines must define the scaled arcs. Each line describes one arc. The description includes 8 numbers: (x), (y) - x- and y-coordinates of the centre of the arc, (r) - the radius in units of x-axis, (f1) and (f2) the start and

final angles in degrees for drawing the arc. The 6-th number (line thickness) determines the line thickness in pt. The 7-th parameter (color) defines the color in a usual manner (section 2.6). The last 8-th parameter defines the vertical scaling the arc. If (line thickness) is not zero then the arc will be drawn as a scaled vertically color line. If (line thickness) equals 0 then the arc will be drawn as a scaled filled region.

Once again an example of using this graphical object is shown below

```
  kind=5 xco=80 sca=1 yco=80 xas=250 fas=250
  xbs=1 xts=1 fls=1 frs=1 sdi=0 gxs=0 gfs=0
    dfile=here
2 1 1 1
5 9900 region 1
42.272 0. 149.875 25. 187. 25. 187. 22.698 89.306 0.
4 999900 region 2
0. 6.33 187. 6.33 187. 14.062 0. 14.062
4 2. 990000 line 1
30 18 50 22 70 18 90 22
120 1 99 'is1 (Victory)H(index)it ds1'
110 10 20 0 360 5 99 0.5 arc 1
%eof
#xaxis 0 187 0 50 4
#faxis 0 25 0 5 4
  xtitle=(Energy (meV))Lc
  ftitle=(Angle \()Lc(m)G(rad\))L
  txx=0 txy=0 tfx=0 tfy=0 col=0
#draw
```

One may see the lines which define the parameters of regions, lines, texts and arcs may have comments after the total information is presented. The figure of this program looks like this.



Sorry, this graphical object is not well done, the structure of data is rather rigid. However it is very useful. The difficulty arises only on the first stage of the work. With an experience the inconvenience decreases. It is necessary to note that the user must be careful in preparing the data file because some errors in data cannot be improved and these lead to a fatal end of working the program. The reason is that the parameters are defined directly and have no the default values.

4. Rich text with figures

Rich text with figures is the powerful graphical object of (**kind = 6**). There are several problems in creating the beautiful text using simple text editors. For example, for russian users it is inconvenient to write the russian letters directly in postscript because the russian symbols are coded by latin symbols and one needs to remember the table of correspondence. In VKPS russian font **cyr30fnt.psf** this table is simple enough and it contains letters close in sound to english letters, for example, a -> a, p -> r, c -> s and so on. However, for some letters the correspondence is not so evident. The correspondence may be obtained each time by comparison the data file of initial text with the text of resulting ps-file.

Second problem is formatting the text when it is necessary to place the text inside the columns of definite width with even boundaries or to change some properties like a spacing between lines, a font, a size and so on. A separate and very important task is to combine the text with figures.

To make such a work to be comfortable for users the VKPS is able to translate automatically the simple ASCII text to a direct postscript fragment. The initial (data) text may be written using any simple text editor with Windows coding of russian symbols. The text may have the words which represent thoughts of the author together with additional commands which rule how to represent the text itself in the document. In analogy with other editors we will call it as a rich text. One may use Windows editor, for example, Notepad for writing the source text in the data file.

Afterwards the rich text will be transformed to the postsript fragment for a moment. In the process of compilation the fragment will be scaled according to the current value of parameter SCA and placed by **left-top** corner at the point (XCO, YCO) at the page.

The text may be large and may contain a lot of

figures so that it may occupy a lot of pages. As an example one may examine this document which is prepared just with VKPS program completely. The program VKPS makes a passage to the next page automatically when the vertical position of the next line becomes lower that the bottom margin parameter **bmp**. The new position of line in the next page will be a top boundary of A4 format page minus the value of top margin parameter **tmp**. In principle, a whole book may be described as one VKPS graphical object. On the other hand, VKPS is able to translate different fragments step by step. Therefore it may be more convenient to consider the first chapter in one data file, the second chapter in another data file and so on.

The data file of this graphical object is a simple ASCII file containing the source text with some driver symbols which have a meaning of commands. Such a data file will be named (**xxx.rt**) file contrary to (xxx.vkp) file of input stream of VKPS program. In the resulting text the russian font is similar to Helvetica or Arial fonts while latin symbols in this text by default is drawn by Helvetica font. All standard postscript fonts are reachable as well as many sizes. The module uses the parameters:

dfile - to specify the name of data file (xxx.rt),
lhe - to specify the height of lines in pt,
lwi - to specify the width of column in pt,
bmp - to specify the bottom margin in pt,
tmp - to specify the top margin in pt,
str - to specify the margin between columns,
mod - to specify the modification.

For the sake of simplicity to deal with the rich text it may contain the comment lines similarly to the input stream of VKPS. The comment lines have to contain the symbol (%) in the first position.

However there are two special comments which are used to specify the extra format of each page like header, footer, page numbering, some additional title of the page etc. These comments are used only in the case when the page is full and it is necessary to pass on the next page. The first comment looks like this

```
%pagen 34
```

where (%pagen) is the name of comment and (34) is an argument. This comment specifies what number will be placed on the current page. In an example presented it is 34. If the same comment with other argument will be absent then the next page will have number incresing by one (35) and so on. Nevertheless it is not making automatically. The second comment is necessary which specifies the position of the page number together with all other information about the page frame. The second comment has a structure like this

```
%paget text on first line of pagetitle PS program\
  'text of second line of pagetitle PS program\'
  ' and so on somewhere with (???)L\'
  'last line of pagetitle PS program'
```

Here (%paget) is the name of comment and all other text is an argument. The argument is a direct PS program which specifies the text of header or footer or frame or some other feature which must be put on each page of the document if the new comment will not appear which redefines the program. The structure of the comment is as follows. The first line contains the text just after the name in free format. If the continuation is necessary then the last symbol of the line must be (\). The second line must be in apostrophes. Once again if the continuation is necessary the last symbol inside the apostrophes must be (\). And so on. The program may contain in some place three symbols "?" together. Such a combination will be replaced on the current page number as was specified by the comment (%pagen). For example the rich text file of this document has the next comments:

```
%pagen 1
%paget gs 1 0 0 srgb 40 790 m ds1(Victor Kohn)I \
  'is1 297 790 m ds1\'
  '(Introduction to VKPS and PostScript)Lc\'
  ' is1 555 790 m ds1(???)Le is1 0.25 slw \'
  ' 40 783 555 783 ls gr'
```

The total length of the PS program is limited by 512 symbols.

4.1 Modifications of text formatting

The parameter **mod** allows to make different ways of formatting the text. Normally (SCA = 1) the size of letters is 14 pt. However, the user may choose an another size through the parameter **sca** which allows to scale the whole fragment. This means that not only the size of letters will be changed but also the width of lines (column) and spacing between lines. In other words, the total graphical object will be scaled as a whole. This case is realized when **mod = 0, 2**. On the contrary, when **mod = 1, 3** the basic size of letters will be scaled only but the width and the height of lines stay the same as without a

scaling as well as the total size of graphical object. For example, with (sca = 0.85) one will obtain the size of 12 pt. It is only the basic size of letters. An additional change of size can be ruled by commands of rich text. As for figures which are included inside the text they have always the size which was pointed out in its reference (see below).

Another modifications concern the kind of line breaking. When **(mod = 0, 1)** the text fills a column up to the end of paragraph and the ends of lines in the source rich text mean nothing completely. In some formatting editors (LaTeX, Postscript) the end of line is equivalent to the blank sign. To exclude this rule in Postscript one need to use the backslash sign (\) as a last sign of line. In VKPS the end of line means nothing originally. If one want to introduce a blank sign at the end then it must be done in explicit form. It is convenient to have the first symbol of each line as a blank sign. The end of paragraph is specified by an empty line or a special command \Ex (see below). On the other hand, there are four standard fillings of column: with even left side, even right side, centered lines with fixed space symbol width and even both left and right side with different space symbol width. Few blank symbols in the source file mean the same as one blank sign (similarly to LaTeX).

When **(mod = 2, 3)** the program works differently, namely, the end of each line is the end of paragraph and each space symbol is taken into account with a fixed width. When the column is wide enough a complete correspondence will arise between initial ASCII text and formatted text. However, if the column is narrow the lines will be broken.

Let us resume the possible modifications:

mod = 0, 2
the size of letters, the width of lines and the line spacing are scaled simultaneously, originally the letters are of 14 pt size. The size of whole fragment is changed.

mod = 1, 3
the size of letters is scaled from the size of 14 pt while the line width and height stay the same together with the size of whole fragment.

mod = 0, 1
the ends of lines mean nothing, the few blank symbols means the same as one sign. The empty line or a special sign \Ex specify the end of paragraph.

mod = 2, 3
the end of line means the end of paragraph, each blank symbol is taken into account in explicit form with a fixed size.

The formatting procedure is performed by the program VKPS. In the case of standard postscript fonts this procedure may be device dependent if the characters of postscript fonts have different properties on different devices. Usually it is not a case and difference may arise only between the interpreters of different levels. At least one cannot find the difference between Ghostview 4.01 for PC and network printers. The program VKPS has an information about the size of each letter in each font among the recommended to use. Other fonts can be used also but the right side of column will be approximately even only.

In principle, postscript-language allows to perform the line breaking algorithm directly and this way is device independent. However, it is more convenient to use some simple rich text with non-postscript commands and non-postscript figures in the source file. Afterwards all necessary calculations are performed by the program VKPS together with an automatic formatting the text on pages.

The problem here is that the program GhostView allows to see the pages in arbitrary order and it uses for this purpose the comment lines like this

%%Page: 3 3 .

It is impossible to write this comment line to the postscript file within the postscript procedure of the same file. At least I cannot make this.

In (mode = 0, 1) few blank (space) signs inside the rich text mean the same as one blank sign because the width of space between words is defined automatically.

4.2 Commands of rich text

The commands of rich text begins from the symbol (\) (backslash) followed by one small or capital letter and in some cases by numerical parameter. The blank sign before the command only is significant. The blank sign after the command means nothing and may be used to make the text more readable. Only one exclusion is made for the command **\m**. The blank sign after this command is significant. The same small and capital letters mean different commands. When one needs to put the sign (\) itself in the text it can be done as command (\\). Each command opens one new feature which will arise in subsequent text and will work up to the end or

when a new command cancels the influence of this command. For example, each new set of font cancels the preceding set. The same for sizes and levels. Below all commands are shown in table together with explanation separated by == and an example:

\\ == symbol \

\R == russian font.
source: \R раз one два two три three четыре four
result: раз one два two три three четыре four
This font is absent in a set of standard postsript fonts. To use the font it is necessary to specify the font by means of line
hfile=cyr30fnt.psf
in (xxx.vkp) (input file) before a call of the module. The file **cyr30fnt.psf** is distributed together with the program VKPS.

\H == standard postscript Helvetica font.
source: \H one two three four five
result: one two three four five
This font is used also for nonletter characters of russian font like . , : ; ( ) 0 1-9 [ ] { } and so on. Moreover when a russian font is specified and latin symbols arises in text this font is used automatically.

\L == standard postscript Palatino-Roman font.
source: \L one two three four five
result: one two three four five
You see the latin font is a base font of this documentd.

\B == standard postscript Palatino-Bold font.
source: \B one two three four five
result: **one two three four five**

\I == standard postscript Palatino-Italic font.
source: \I one two three four five
result: *one two three four five*

\G == standard postscript Symbol font
source: \G a b g d
result: α β γ δ

\m == middle level of text in line (default).
This command is used to finish a writing of subscript or superscript. The blank sign before this comand will be scaled that is inconvenient in many cases. That is why the blank sign after this commmand is significant and this rule is valid only for this command.

\u == up level of text (for superscript).

\U == up more level of text (for superscript in superscript).

\d == down level of text (for subscript).

\D == down more level of text (for subscript in subscript).

\r == return.
This command is used for writing a subscript and a superscript simultaneously one below another. This may be done as \d subscript\r\u superscript. The longer index must be the second.

Below an example of using these commands is shown
source: \H A\s\G\d abg\r\u stu\m\n\m - \H S\s\d abc\m\n
result: $A^{\sigma\tau\upsilon}_{\alpha\beta\gamma} - S_{abc}$

\n == normal size of text (default).

\s == small size of text (0.8 scaled)

\f == footnote size of text (0.7 scaled)

\t == tiny size of text (0.55 scaled)

\l == large size of text (1.2 scaled)

\b == big size of text (1.5 scaled)

\h == huge size of text (2. scaled)

\e == enormous size of text (3. scaled)

Below all sizes are shown together with the command \Xnnn, \Ynnn (see later)
source: \t tiny,\X10\f footnote,\X10\s small, \X10\n normal,\X10\l large,\X10\b big, \Y12\El \h huge,\X10\e enormous.
result: tiny, footnote, small, normal, large, big, huge, enormous.

The following commands explain the paragraph formatting.

\Ex == end of text paragraph. Here the second sign is an argument. The argument has four values: **x == w, l, c, r .** The purpose of argument is to set the kind of subsequent paragraphs:
\Ew -- even left and right sides,
\El -- left aligned (hard blank sign),
\Ec -- centre aligned (hard blank sign),
\Er -- right aligned (hard blank sign).
The argument may be absent. Then preceding setting is used.

\Xnnn == to put a hard horisontal space in a line. The width of space is defined by argument as a positive integer number in pt (one, two or three digits). For example, \X10 means 10 pt hard space. Contrary to the blank sign which is scaled together

with symbols by commands \s \f and others, the hard space is independent of scaling the fonts.

\Ynnn == to put a hard vertical space between the lines. The depth of space is defined by argument as a positive integer number in pt (one, two or three digits). Contrary to command (\Xnnn) which always moves the text inside the line to the right the command (\Ynnn) may move the line down or up. Namely, if the argument nnn < 500 then the next line is moved down. If nnn > 500 then the difference (nnn - 500) is the amount of vertical space in pt for moving the line up. This property may be used to place the part of text at the left, part at the centre and part at the right of the same line. For example, if (lhe = 14) and (mode = 1) then the rich text: . . .\El abcd\Y514\Ec efgh\Y514\Er ijkl\Ew . . . will lead to the line

absd efgh ijkl

One has to remember that the command \Ynnn must be used with some restrictions. Due to the algorithm used in VKPS interpreter it is recommended to use the vertical space only at the end of paragraph together with \E command. Normally the combinations \Y20\Ew and \Ew\Y20 mean the same. However, if the text goes through the region with a figure the first combination is better and more stable. Before the command of figure the only first (\Y20\Ew) combination will give a correct result when both top side of figure and new line of text have the same position. Naturally other situations are also possible but the quality will be on a user taste.

\P == hard end of page.
This command allows to begin next column or next page if the next column is impossible even if the current column is not full. Before the usage of the command the paragraph must be finished by \E command.

\Oxxx: X Y k n == to put a figure.
Contrary to all other commands this command is rather complicated and needs in additional explanation. The command \O may arise only after end of paragraph \Ex and the command \Ex must have an argument in explicit form. The command \O must be put in a separate line from the first position. All symbols between (O) and (:) are ignored (it is a comment) but after the sign (:) the line must contain four numbers: (X, Y, k, n). The number (X) means a horisontal size, the number (Y) - vertical size of the rectangular region in pt in which the figure will be placed just instead of this command. The number (k) means the key which has one of three values (-1, 0, 1) and means the position of graphical object at the left side, the centre and the right side of the column.

When (k = -1, 1) the text will go through the column together with a figure if the rest of column allows to make this. When k = 0 the text will jump through the figure. The position of figure is set automatically just below the paragraph which precedes the command \O. The number (n) means the number of different descriptions of the figure, namely, the number of the commands #draw in the input file (xxx.vkp, see below). In a current version of the program VKPS there is no checking on a position of figure inside the page. All figures are drawn at the current page even if their bottom side turns out to be lower than a bottom of the page. The user must be careful about this. If the place of the figure is not well defined then it would be changed by user.
The command looks like this

```
. . . some text of source file . . .\Ew
\Object-1 : 250 250 3 2
. . . some text of source file . . .
```

The command \O only reserves the region for the figure but the figure itself is not described in data (xxx.rt) file. The real description of the figure as a standard graphical objects of VKPS program must be placed to input (xxx.vkp) file just after a call of the rich text procedure. Then the first n commands #draw will define first object, referenced in the rich text with number (n), second n commands #draw - second object and so on. The objects may have all kinds except kind = 6. When drawing these objects the parameters **xco, yco** are not used. Instead of this the figures are placed into the reserved region. Then the scaling of figures is performed automatically according to the formulas:

```
  sca = min( s1, s2 ),
  s2 = ( Y - 20 )/( fas + 50 ),
  s1 = ( X - 20 )/( xas + 60 )
```

in the cases of kind = 1, 2, 4, 5

```
  s1 = ( X - 20 )/( 70 + xas*( 1 +
     + (xlp-xfp)*nfu*hsf/( (nxp-1)*(xen-xbe) ) ) )
```

in cases of (kind = 3)
where all values are in pt and X, Y are the sizes of reserved region for the figure. In this way the figures always appear in a centre of selected region.

One have to remember that the scaling change the size of figures and symbols of titles at the axes. Therefore to have a better view of these parts of the figure the user must choose the sizes of axes **xas** and **fas** as necessary. The experience will come during the work.

In the case of object of kind = 0 only the translation to the left-bottom corner of the region is performed while the scaling is made according to the current value of the **sca** parameter. The user must be careful to set the graphical object inside the region by proper choice the translation and scaling inside the postscript text of object.

The program is able to format a large text which needs few pages. When the column is filled in, i.e. a position of the next line becomes lower than the bottom boundary of the text on the page the program verifies a possibility to place the second column with the space between columns defined by parameter **str**. The A4 format of page is expected. If the width of column is not large and there is a place for the second column then VKPS begins new column with the same vertical position as the first column. The same for the third column and so on. When the next column has no place the current page is finished and new page is opened. Hence the multicolumn regime is assumed automatically. When it is undesirable the user may use the wide column and empty graphical object which takes the extra space.

BUGS:

The program was tested in different regimes and some bugs were discovered which reasons I cannot understand to this moment:

1. It is not forbidden but I don't recommend to use the command \E (without an argument) at ends of line in (xxx.rt) file. However, it is often convenient to have a correspondence between the end of lines of (xxx.rt) file and printed copy. If one likes to use this then it is desirable to have this command as a last command of xxx.rt file line and to move all other commands on a next line. Namely,
...some text...\Er\B
...some text...
may give bad result in a sense that a right side will be shifted. The same in the form: ...some text...\Er
\B...some text...
will give a correct result.

2. I have found that sometimes, for example, when two lines of (xxx.rt) file correspond compelety to the printed lines in the column, the right side of line becomes shifted from the position which it must have. To avoid this one can change the width of lines in (xxx.rt) file (simply break one line on two lines). This arises very seldom and looks like a mystery.

Appendix 1. Table of VKPS parameters

Here all the parameters are presented by shortened name together with a description and the initial value.

psf (postscript file name) [0001.ps]
hfi (head file name) [head30.psf]
cfi (color file name) [col30vk.psf]
dfi (data file name) [vksign.dat]
xti (x-axis title) [(axis X)Lc]
fti (f-axis title) [(axis F)Lc]
yti (y-axis title) [(axis Y)Lc]

kin (kind of graphical object, integer 0-6) [1]
xco (x-coordinate of object position in pt) [70]
yco (y-coordinate of object position in pt) [270]
sca (scaling factor for object scaling) [1]
xas (x-axis size of figure in pt before scaling) [350]
fas (f-axis size of figure in pt before scaling) [300]

xbe (x-axis begin value) [0]
xen (x-axis end value) [105]
xfm (x-axis first long mark, having value) [0]
xsm (x-axis step to next long mark) [10]
xns (number of short marks between long) [4]

fbe (f-axis begin value) [0]
fen (f-axis end value) [75]
ffm (f-axis first long mark, having value) [0]
fsm (f-axis step to next long mark) [10]
fns (number of short marks between long) [4]

xfp (x-argument, first point) [0]
xlp (x-argument, last point) [1]
nxp (number of x-arguments) [100]
nfu (number of functions to be drawn) [1]
iff (index of first function in the file) [1]
nmx (number of dots in the line of matrix) [100]
ifx (index of first dot of line to be drawn) [1]

yff (y-axis, first function for a long mark) [1]
ysf (y-axis, step to number of next function) [1]
yfm (y-axis, value at first long mark) [0]
ysm (y-axis, step to value of next mark) [1]

xbs (x-bottom axis switch, 1 - yes, 0 - no) [1]
xts (x-top axis switch, 1 - yes, 0 - no) [1]

fls (f-left axis switch, 1 - yes, 0 - no) [1]
frs (f-right axis switch, 1 - yes, 0 - no) [1]

fun (a unit for functions) [1]
mas (marker size (radius of circle) in pt) [0]
lth (line thickness in pt) [0.5]

sdi (a direction of strokes at marks on axes) [0]
gxs (grid of x axis switch, 1 - yes, 0 - no) [0]
gfs (grid of f axis switch, 1 - yes, 0 - no) [0]

vsf (vertical shift of function in scale of f-axis) [0]
hsf (horisontal shift of function) [0]
vsw (visibility switch, 1 - no, 0 - yes) [0]

txx (x-axis, x-coord. of title shift in pt) [0]
txy (x-axis, y-coord. of title shift in pt) [0]
tfx (f-axis, x-coord. of title shift in pt) [0]
tfy (f-axis, y-coord. of title shift in pt) [0]
tyx (y-axis, x-coord. of title shift in pt) [0]
tyy (y-axis, y-coord. of title shift in pt) [0]

lhe (line height in pt) [20]
lwi (line width (colomn) in pt) [450]
tmp (top margin of page in pt) [70]
bmp (bottom margin of page in pt) [70]
str (strip between columns) [15]
mod (modification) [0]
col (color of axes and general) [0]
lco (color of lines in the figure) [0]

Appendix 2. Postscipt commands of head files

This appendix presents the postscript procedures which are defined in the head files. The version 3.0 allows one to use many head files - up to 10. The necessary head file having the name (head30.psf) contains the PS commands which are used by VKPS itself. This file must exist on the current directory of the program or the total path of the file must be specified in the opposite case. This file cannot be rewritten. Another head file (cyr30fnt.psf) describes the russian font for using in the title and rich text. This file must exist and be specified when the russian letters are used. Other head files may be arbitrary and may be rewritten by user who knows the postscript language. These may contain the procedures which will be used in the direct postscript fragments of the (xxx.vkp) file or postscript data file. Here each command is presented in a pure form with some explanation and example for a usage.

A2.1 Head file **head30.psf**

bd (bind def) help operator

ed (exch def) restore the parameter in procedure (1 argument), usage: {/X ed . . .}

gs (gsave) save the graphical parameters which will be used later after restoring

gr (grestore) restore the graphical parameters which were saved by **gs** operator

tr (translate) to translate the coordinates (2 arguments), usage: 20 30 tr

sca (scale) to scaled the coordinates (2 arguments), usage: 0.5 2 sca

np (newpath) to open new path

m (np moveto) to begin new line (2 arguments),usage: 20 50 m

l (lineto) to draw the line segment from current point to the new point (2 arguments), usage: 30 55 l

sl (stroke) to show the path as a line on the page

ls (m l sl) to show the straight line segment (4 arguments), usage: 200 225 250 300 mls

c (curveto) to draw the smooth curve line which is obtained as cubic spline from the current cursor position to the point X3,Y3. The points X1,Y1 and X2,Y2 are not on the curve and these are used for the interpolation; (6 arguments), usage: 0 0 m 20 50 100 200 20 350 c

rl (rlineto) to draw the line from current point to the new point which is shifted from the current point by the vector X,Y (2 arguments), usage: X Y rl

rm (rmoveto) to specify the new cursor position which is shifted from the current position by the vector X,Y (2 arguments), usage: X Y rm

slw (setlinewidth) to set a line width in pt

r (rotate) to specify the angle of rotation in degree of all following graphical objects (1 argument), usage: 90 r

srgb (setrgbcolow) to set a color (3 argument, Red Green Blue components from 0 to 1), usage 0.5 1 0.3 srgb

sg (setgray) to set a gray level (1 argument from 0 to 1), usage: 0.5 sg

is1 (1.25 1.25 sca) standard 1 increasing scaling

is2 (1.5625 1.5625 sca) standard 2 increasing scaling

is3 (2 2 sca) standard 3 increasing scaling

is4 (3.125 3.125 sca) standard 4 increasing scaling

is5 (4 4 sca) standard 5 increasing scaling

ds1 (0.8 0.8 sca) standard 1 decreasing scaling

ds2 (0.64 0.64 sca) standard 2 decreasing scaling

ds3 (0.5 0.5 sca) standard 3 decreasing scaling

ds4 (0.32 0.32 sca) standard 4 decreasing scaling

ds5 (0.25 0.25 sca) standard 5 decreasing scaling

s (show) to show the text by current font with the left boundary at the current point (1 argument - the text itself in round brackets), usage: (abracadabra)s

sp (stringwidth pop) to remember the width of text

sc (dup sp -0.5 mul 0 rm s) to show the text with the middle at the current position (1 argument - the text itself in round brackets), usage: (abracadabra)sc

se (dup sp -1 mul 0 rm s) to show the text with the end at the current position (1 argument - the text itself in round brackets), usage: (abracadabra)se

fir (closepath gs fill gr) to close the path defined previosly and to fill the region inside the path by current color, usage: 0 0 m 0 2 rl 2 0 rl fir

fr (sg fir) to close the path defined before and to fill the region by grey color according to the gray level (1 argument - grey level), usage: 0 0 m 0 2 rl 2 0 rl 0.5 fr

f (findfont exch scalefont setfont) to specify the size in pt and kind of new font which becomes a current font (2 arguments), usage: 14 /Palatino-Roman f

PR (/Palatino-Roman f) to specify the standard font (1 argument), usage: 14 PR

PB (/Palatino-Bold f) to specify the standard font (1 argument), usage: 14 PB

PI (/Palatino-Italic f) to specify the standard font (1 argument), usage: 14 PI

Sy (/Symbol f) to specify the standard font (1 argument), usage: 14 Sy

He (/Helvetica f) to specify the standard font (1 argument), usage: 14 He

L (14 PR s) to show the text by standard font and size (1 argument - the text itself in round brackets), usage: (..text..)L. Subsequent texts of the same font may be shown by (..)s

Lc (14 PR sc) to show the text by a middle point at a current position by standard font and size (1 argument), usage as in preceding command.

Le (14 PR se) to show the text by end at a current position by standard font and size (1 argument), usage as in preceding command.

B (14 PB s) to show the text by bold font (usage see above)

I (14 PI s) to show the text by italic font (usage see above)

G (14 Sy s) to show the text by symbol font (usage see above)

H (14 He s) to show the text by helvetica font (usage see above)

ind (/X ed /Y ed /tt ed 1 Y rm X X sca tt s /X 1 X div def X X sca 1 Y neg rm) to shows shifted and scaled text as index (3 arguments - text itsef in round brackets, vertical shift Y of the text relative to the current position and scaling X), usage: (text) -4 0.6 ind

ib (-4 0.6 ind) to show the subscript of the text of standard font

it (5 0.6 ind) to show the superscript of the text of standard font

lfra (/Ys ed /Xs ed /Y ed /X ed X Y m Xs 0 rl 0 Ys rl Xs neg 0 rl 0 Ys neg rl sl) to draw the line frame with X,Y point as a position of left-bottom corner and Xs,Ys as sizes of width and height (4 arguments), usage: X,Y,Xs,Ys lfra

Sh (0 rm) to make horizontal relative move of the current cursor position (1 argument), usage: 100 Sh

Sv (/Y ed 0 Y rm) to make vertical relative move of the current cursor position (1 argument), usage: 100 Sv

A2.2 Head file **cyr30fnt.psf**

Rf (12 slw 1 setlinejoin 2 setlinecap /Cyr30vk f) to specify a standard russian font (1 argument), usage: 14 Rf

R (14 Rf 2 0 rm s -1 0 rm) to show the text by standard russian font (1 argument), usage: (..text..)R

A2.3 Head file **head-a.psf**

The commands of this file are not used by VKPS. Therefore the number of command in this file may be increased or descreased or rewritten by user. This commands may be used in direct postscript

programs in both VKPS and pure PSL file.

Arr (/Xs ed /Xr ed /Y ed /X ed gs X Y m Xr r Xs Xs scale -2 -7 rl 4 0 rl -2 7 rl 0 gr gr) to draw the arrow head with a needle at the point X,Y rotated on the andle Xr (degree) from the vertical position and scaled in Xs times (4 arguments), usage: 100 200 -90 1.5 Arr

Arl (/Xr ed /Xl ed /Y ed /X ed gs X Y tr Xr r 0 0 Xl 0 mls 0 0 90 1 Arr Xl 0 -90 1 Arr gr) to draw the straight line with the arrows on two ends (4 arguments: X,Y - the position of basic end, Xl - the length of line and Xr - the angle of rotating in degree from the horisontal arrangement), usage: 100 200 100 -90 Arl

ffra (/Xf ed /Ys ed /Xs ed /Y ed /X ed X Y m Xs 0 rl 0 Ys rl Xs neg 0 rl 0 Ys neg rl Xf fr) to draw the filled by grey rectangular region (5 arguments: X,Y position of the left bottom corner, the x,y sizes and gray level), usage: 0 0 20 10 0.5 ffra

rfra (/Xr ed /Ys ed /Xs ed Xr r Xs 0 rl 0 Ys rl Xs neg 0 rl 0 Ys neg rl sl Xr neg r) to draw the rotating line frame (3 arguments: Xs, Ys as width and height and Xr (in degree) as an angle of rotating around the left-bottom corner which is placed at the current position), usage: 100 200 30 rfra

tfra (/Xf ed /Xd ed /Yp ed /Xp ed /Y ed /X ed X Y m Xp 0 rl 0 Yp rl Xp neg 0 rl 0 Yp neg rl sl /Y Y Yp add def /Yd Xd neg def X Y m Xd Xd rl Xp 0 rl 0 Yp neg rl Yd Yd rl 0 Yp rl Xp neg 0 rl Xf fr) to draw the line frame with a gray 3D-space border to make the text inside the frame to be brightly selected (6 arguments: X,Y - position of the left-bottom corner, Xp,Yp - width and height of the frame, Xd - the size of border, Xg - the gray level), usage: 100 250 200 50 10 0.5 tfra

AA ((A)L gs -7 10.5 rm 0.5 0.5 sca (o)s gr) Angstroem unit

hh ((h)L gs -9.8 3.9 rm (-)G gr) Plank constant

int (0 5 rm (\363)G -11 -10 rm (\365)s 0 5 rm) integral sign

frac (/Xn ed gs 4 6 rm X gr gs 4 -9 rm Y gr 14 Sy Xn{(\276)s}repeat) to draw the fraction in formulas (3 arguments: the numerator X, the denominator Y and the number of signs for a middle line, X and Y are the procedures which must be defined previously), usage: /X{ ... }bd /Y{ ... }bd 10 frac

stb (/Y ed 0 Y neg rm gs) to make the current line position on Y pt below the preceding line position (1 argument), the text itself must be defined separately by a lot of operators if necessary and must be closed by operator 'gr', usage: 20 stb ..text.. gr

st (/tt ed /Y ed 0 Y neg rm gs tt s gr) to show the text line by current font, the position of beginning is shifted on Y pt down from the current position, the current position is placed at the beginning of new text (2 arguments: Y and the text itself in round brackets), the operator is used for writing the large text of many lines with even left boundary as direct PS program, usage: 20(..text..)st

blb (is3 0 -3 rm (\()L s 0 3 rm ds3) big left round bracket

brb (( )L is3 0 -3 rm (\))s 0 3 rm ds3) big right round bracket

min1 (-6 13 rm (_)L 0 -13 rm) to draw the minus sign above the high figure as it is accepted in Crystallography

min2 (-6 10 rm (_)L 0 -10 rm) to draw the minus sign above the low figure as it is accepted in Crystallography

lgay (/Xm ed /Xn ed /Y1 ed /X1 ed /Y ed /X ed /Yd Y1 Y sub def Xn {/Yc Yd Xm log mul Y add def X Yc X1 Yc mls /Xm Xm 1 add def}repeat) to draw the logarithmic short marks set between long marks on the f-axis (6 arguments: X,Y - absolute left position of the bottom long mark in pt, X1,Y1 - absolute right position of the top long mark in pt, Xn - number of marks to draw, Xm - number of first mark), explanation: decimal logarithm scale has long mark at each integer number n for a logarithm that corresponds 10^n , therefore the short marks corresponds to log/10/ 2,3,4,5,6,7,8,9, however at the beginning or end of axis not all marks will be presented, therefore the procedure allows to spesify the first mark and number of marks to draw, usage: 90 100 100 300 8 2 lgay

lgax (/Xm ed /Xn ed /Y1 ed /X1 ed /Y ed /X ed /Xd X1 X sub def Xn {/Xc Xd Xm log mul X add def Xc Y Xc Y1 mls /Xm Xm 1 add def} repeat) the same as preceding command but for X-axis

Appendix 3. Introduction to PostScript

PostScript language is a device independent language

of low-level. This means that it has a lot of operators (400) each of them makes some privitive actions and many operators are needed to perform the whole work of description of document which has to be printed or displayed. The operators give the user all possibilities to manipulate the numerical data and strings of symbols with different fonts. The total description of all operators is in the book **PostScript Language Reference Manual (second edition), Addison-Wesley Publishing Company, Inc., 1991** which has above than 700 pages. Therefore the total description is impossible here and it is out of interest. Here only the main features are presented which will be used later.

In principle, each PS file is a program which must be fulfilled by some interpreting device automatically from the beginning to the end. Therefore all parameters must be well defined and all problems must be solved at the preceding stage. When the program stays in its turn to be fullfilled by network printer any interaction with its creator becomes impossible. A device independence allows one to use a usual text of latin symbols to write the program.

PostScript programs must be written as an ASCII-text with a use of only first 127 ASCII symbols, therefore these can be transfered by means of simple e-mail connection between several computers. PostScript language (PSL) was introduced first in UNIX operation system and it is used by network printers as an input stream of information for printing. PSL commands are fulfilled directly, without previous compiling and checking for errors. Therefore any program-interpreter is enough for performing actions described in PSL.

A structure of the language is rather simple at the first stage as well as rather complicated when a usage of all possibilities is desirable. The numerical data can be written in free format as well as commands. Any number of blank symbols as well as symbols of end of line are used as a separator between data or commands. A symbol % or %%%% is used to point out that all following text up to the end of line is a comment and it is not performed. Nevertheless, some comments have a special structure and bring to interpreter a helpful information about a document. This is concerned, first of all, a beginning of the document. For example, a multipage postscript document has to have a structure like this:

```
%!PS-Adobe-3.0
%%Title: VKPS 3.0 tutorial
%%Creator: VKPS 3.0 (August 2000)
%%Copyright: Victor Kohn (Kurchatov Institute)
%%Document fonts: Palatino-Roman Symbol
%%BoundingBox: 0 0 596 843
%%Pages: 2 1
%%EndComments
%%BeginProcSet
    ... text of procedures for all pages
%%EndProcSet
%%EndProlog
%%Page: 1 1
    ... contents of first page
showpage
%%Page: 2 2
    ... contents of second page
showpage
%%Trailer
%%EOF
```

Here the first line declares a PostScript file. The lines 2-5 are not necessary but these are useful for readers. The line 6 (BoundingBox:) shows a size of page in units **pt** (point). These units are usual in Postscript language.

1 pt = 0.353 mm
A4 paper has size 210*297 mm = 596*843 pt
1 inch = 2.54 cm = 72 pt

Hence, you see that the line 6 sets a A4 format of page. The line 7 shows what number of pages the document has. A second number is not necessary, usually it points out a number which will be set in a first page but this is not performed automatically. The following lines are clear. These are the frame of the place containing the definitions (procedures) for all pages and frames for each page. The part before line (%%EndProlog) is named "Prologue" whereas the part after this line with pages is named "Script". The prologue contains some preliminary information for the interpreter which used later. Script part describes the contents of real pages and use the information from prologue part for making the text to be clear and shorten. Such a structure is necessary for a description of whole document which contains, for example, figure captions and all figures for some article in different pages, one figure on one page with a lot of free place.

So called Encapsulated Postscript document has another frame. This document describes only one picture which may have finite size (not a full page) and may be used in other postscript files and different (LaTeX, Word) documents with an

additional transformations (scaling, rotation, translation) as a whole. The frame of EPS file is as follows:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 100 100 410 355
%%EndComments
%%EndProlog
  gsave
   ... contents of picture
  showpage
  grestore
%%Trailer
%%EOF
```

Here some postscript commands are also shown which allows to show a document as a whole picture but these are not necessary in EPS file if this document is a part of another document. Here a line (%%BoundingBox) must define a region of the page with any coordinates of left bottom corner (100 100) and right top corner (410 355). The coordinates of all points of the picture must be in these limits.

A3.1 Memory and command interaction

Usually commands for their work must be accompanied by some numerical data. This information may be taken from some accumulator which uses a piece of memory. In compilled languages for this purpose one defines arrays of different kinds and then uses the elements of these arrays. It is convenient for a programmer but is not convenient for a computer because a computer makes a work to find a real element of memory by means of array definition. In interpreted languages different ways are used usually. The Postscript language is one of them.

There are no arrays and other definitions of memory in a pure sense. In principle, the arrays and strings of symbols (text) exist as a structures. However a memory is used without definitions. All memory may be imagined as a very large volume with one door. When the volume is full then only one principle of working is seen:
LAST ENTERED, FIRST WILL LEAVE,
shorter,
LAST IN, FIRST OUT.
This principle is called as a stack-oriented memory usage. It means that when a number arises in a text of postscript program its value is kept in the stack at last position. After an appearance of next number first number takes second position from the end and last number has last position (end). The program does not know at all about the beginning of the stack.

Such a memory organization gives a possibility to a programmer to define previously a lot of numbers and then to write the commands (operators) which will use these numbers as input information. However, the numbers must be defined PREVIOUSLY and in OPPOSITE ORDER, namely, last numbers for first operator and so on. For example, a numerical information may show a position of graphical cursor on a page (absolute position) or coordinates of moving the cursor from the current position (relative shift). To draw the curve as a complex line containing few straight segments one can use operators:

moveto - to move cursor on the new position which is taken from a stack, first y then x coordinate,

rlineto - to draw a straight segment from a current cursor position to a new position shifted from old one on a vector with coordinates which is taken from a stack, first y then x,

stroke - to finish fragment and to show it.

The program may be as follows:

```
25 45 moveto
10 10 rlineto 10 30 rlineto
10 45 rlineto 10 -60 rlineto
stroke
```

As a result one obtains a curve which passes through the points:
25 45, 35 55, 45 85, 55 130, 65 70
However, the same result can be obtained with a use of cycle operator

n {...} **repeat**

to make n repetitions of the text inside {} as follows:

```
10 -60 10 45 10 30 10 10 25 45
moveto 4 {rlineto} repeat stroke
```

One may choose different principle of programming: the operators step by step with their arguments which precede the operator or first all arguments in reverse order and then all operators. The second way is convenient for an automatical postscript generation by means of some executing program written by user (xxx.exe file in MS-DOS) and usually leads to a shorter text which is, however, difficult to understand.

A3.2 Definitions and procedures

A very convenient for a programming but a very hard for understanding the programs by other men is a possibility to use new definitions. It means that any piece of text of program may be defined as marked one having some name and later a programmer can use only a name of this piece instead of the total text. In such a way any operator may take a new name, any repeated fragment of program can take any name and then is used few times by means of its name. The definition is declared as follows:

/newname {text of operator} bind def

If {text of operator} does not contain other procedures or new operators the "bind" operator may be omitted. Once defined operator (procedure) can be used in other definitions. The head files described above just contain some definitions which are used for postscript generation by VKPS and may be used in direct postscript fragments. As an example one may see the contents of the head files. It is very useful to keep the same definitions in all files which can form the new language instead of original postscript operators. This language may be more convenient for a definite purposes. The simple definitions have no agrument and can be put inside the text directly. Some definitions use previously defined other definitions or parameters. The complicated definitions have arguments which must precede the name of definitions. These definitions are, in reality, the procedures. Since each PS file is ready for execution it contains all it's definitions in a Prologue part of the file and one can find a lot of ready procedures in these files. The examples of fine procedures may be found in the books.

Postscript allows to use many definitions. Each parameter and procedure may be redefined many times. A huge set of definitions may be placed into the dictionary and a number of dictionaries may be as large as necessary. The same definitions may have different sense in different dictionaries. Thanks to the property of Postscript pointed above it is inconvenient to use the original operators of language. Really each programmist has his own set of new procedures for a usage in the main part of postscript file. These procedures are the gold foundation. They are used each time in each new file. Therefore it is convenient to keep them in a separate files and to make a copy each time when necessary. This is a reason of appearance of head files in VKPS.

A3.3 Graphical operators

Among the numerous operators the graphical operators play a more significant role because the postscript language is graphically oriented. That is why in this section a list of most important graphical operators with their short description is presented. Below the operators will be written by small letters while the possible arguments will be as capital letters, a short definition (SD) of head files will be pointed out as well.

newpath - to open new path, the current cursor position will be continued to the first path position if (moveto) is absent.

closepath - to close the current path, the last point will be continued to the first point to make the region.

X Y **moveto** - to move cursor at the point X,Y

DX DY **rmoveto** - to shift cursor on the vector DX,DY; SD=(rm)

X Y **lineto** - straight segment from the current cursor position to the point X,Y; SD=(l)

DX DY **rlineto** - straight segment of length DX,DY from the current cursor position; SD=(rl)

X1 Y1 X2 Y2 X3 Y3 **curveto** - smooth curve which is obtained as cubic spline from the current cursor position to the point X3,Y3. The points X1,Y1 and X2,Y2 are not on the curve and these are used for the interpolation; SD=(c)

X1 Y1 X2 Y2 X3 Y3 **rcurveto** - the same as preceding operator but with relative coordinates

X Y R A1 A2 **arc** - segment of arc as a part of circle with the point of centre X,Y and radius R. The part includes a region between the angles A1 and A2 in degree, the count is unticlock. A1=0, A2=360 define whole circle. One can obtain ellipse by scaling. The operator only defines the part of the path as well as all other operators. To make the real figure additional operators are necessary.

X Y R A1 A2 **arcn** - the same as preceding operator but the counts of angles is clockwise.

X1 Y1 X1 Y2 R **arct** - segment of arc of definite radius R which has as tangent lines the line along the straight segments from the current cursor position to X1 Y1 and from X1 Y1 to X2 Y2

stroke - to close a path and to show it as a line; SD=(sl)

W **setlinewidth** - environment operator which set a width of line SD=(slw)

N **setlinejoin** - to set one of three modifications of drawing a corner between two thick line segments, N=0 - real angle (acute), N=1 smoothed corner, N=2 cut corner

N **setlinecap** - to set one of three modification of drawing an end of thick line segment, N=0 shortened, N=1 smoothed, N=2 longer.

[..] D **setdash** - environment operator which set a level of dash of the line; the argument means: [] 0 - solid line, [3] 0 - punctured line (3on 3off 3on 3off ...), [4 2] 0 - punctured line (4on 2off 4on 2off ...), [3 5] 6 - punctured line ([3+5-6]off 3on 5off 3on 5off ...)

G **setgray** - environment operator which set a level of gray to fill the regions, G=1 is white, G=0 is black, intermediate between 0 and 1 values set the intermediate gray level; SD=(sg)

R G B **setrgbcolor** - environment operator which set a color of lines, regions etc. as a mixture of red, green and blue from 0 to 1. It is good for display and color printers. For black-white printer a black color is obtained by 0 0 0 setrgbcolor (default value), G G G setgrbcolor is equivalent to G setgray; SD=(srgb)

fill - to fill the region inside the current closed path by current gray level or current color

(TEXT) **show** - to show text in round brackets by defined previously size and font; SD=(s)

X Y (TEXT) **ashow** - the same as **show** but in drawing the TEXT each letter will have a space increased by X and Y in two dimensions.

A3.4 Arithmetic operators

Described in head files complex definitions are, in reality, procedures which take their arguments from a stack but don't use them immediately. Instead, they remember them as a definition and use after a work of other operators which can change a stack. Directly this definition are realized by operators **exch** and **def** which renamed in the head file as

/ed{exch def}bd

For example, **25 /X ed** means X=25. More complex text **35 25 /Y ed /X ed** means X=35, Y=25 (don't remember a reverse order). In such a way one can define a variable inside the procedures. The direct definition is **/X 25 def /Y 35 def**. In such a definitions the arithmetic operations:

add - addition,
sub - subtraction,
mul - multiplication,
div - division
idiv - integer divide
neg - negative (Y neg = -Y)

usually work directly on a stack but one has a possibility to define a result of calculation in the same or new variable, for example: **/X X 40 add def** means X=X+40 (X was defined previously) **/Y 0 Y sub def** or **/Y Y neg def** means Y=-Y

There are standard set of functions:

sqrt atan cos sin exp ln log
abs mod floor round truncate ceiling
rand srand rrand

The variables has no special names. The notation of variables, procedures, arrays, text strings and other more complicated structures of memory can be arbitrary.

A3.5 Environment operators

Some operators are necessary to define the parameters of drawing like considering above [] 0 setdash, setlinewidth, setrgbcolor, setgray and others.

There are useful couple of operators which allow first to save a current graphical cursor position and other environment variables and then to restore it many times, namely,

gsave - to save the current graphical state; SD=(gs)

grestore - to restore saved graphical state; SD=(gr)

They are especially necessary when one fills regions because it is difficult to understand a current cursor position after this operation.

Another group of operators allows to redefine all coordinates of the points on a graphic. They are:

X Y **scale** - to scale all x coordinates by X and all y coordinates by Y; SD=(sca)

X Y **translate** - to translate all subsequent coordinates on a vector X,Y; SD=(tr)

A **rotate** - to rotate all coordinates on the angle A in degree relative a current cursor position; SD=(r)

To exclude an action of these operators one has to use the same operators with opposite arguments.

A3.6 Cycle operators

Naturally, there are figures which demand to draw a lot of similar or the same fragments. This is especially concerned the scientific graphics of array of lines through the array of points or array of definite markers. In this case a lot of data represent array of coordinates which determine the positions of the same fragment. Thus, a cycle operator is necessary. Here two kinds of cycle operators are considered among the various such operators existed in PSL. First is simple repetition

n {....} **repeat**

to make n repetitions of the text inside { }. For example, to perform the curve which pass through 21 points one needs to define 42 numbers - the coordinates of the points in a reverse order and to use

m 20 {l} repeat sl

where short definitions of the head files were used. When these points are the results of calculation of some function with a constant step of changing the argument then the values of argument are evident and these may be calculated inside a procedure which is repeated. However there is another way - to use a cycle operator of second kind. A general form of this operator is

P1 DP PN { ... } **for**

to make a repetition of the text inside { } with simultaneous sending one number in a stack before each repetition. First value of number is (P1), last value is (PN), a number increases on a step (DP). Arguments may be numbers or definitions of numbers. The cycle works when a current value of number does not exceed (PN). To use this cycle for our example of drawing a curve of 21 points one can write:

... 21 numbers of function values in reverse order ...
x1 exch m x2 dx x21 {/X ed /Y ed X Y l} for sl

where x1, dx and x21 are the x-coordinate of first point, step and x-coordinate of 21-th point, x2 is for a second point.

A3.7 Creating text by standard fonts

One can see that by means of new definitions there is a possibility to define symbols as pictures and then rescale and moved these pictures creating a text. Nevertheless, as a rule, each postscript interpreter has an information about some standard fonts which can be used directly by means of simple opening of the font. Below the names of these fonts are pointed out in the order of increasing the practical usage:

/Palatino-Roman
/Palatino-Italic
/Palatino-Bold
/Palatino-BoldItalic

/Times-Roman
/Times-Italic
/Times-Bold
/Times-BoldItalic

/Helvetica
/Helvetica-Oblique
/Helvetica-Bold
/Helvetica-BoldOblique
/Helvetica-Narrow
/Helvetica-Narrow-Bold

/Symbol

/Courier
/Courier-Bold
/Courier-Oblique
/Courier-BoldOblique

/AvantGarde-Book
/AvantGarde-BookOblique
/AvantGarde-Demi
/AvantGarde-DemiOblique

/Bookman-Light
/Bookman-LightItalic
/Bookman-Demi
/Bookman-DemiItalic

/NewCenturySchlbk-Roman
/NewCenturySchlbk-Italic
/NewCenturySchlbk-Bold
/NewCenturySchlbk-BoldItalic

/ZapfDingbats
/ZapfChancery
/ZapfChancery-Oblique
/ZapfChancery-Bold
/ZapfChancery-MediumItalic

To open the font of definite size VKPS uses the procedure
/f {findfont exch scalefont setfont} bd
which has two arguments: the size in pt and kind of the font, for example: 14 /Palatino-Roman f

When some font is opened then the text may be defined by different ways. Some symbols may be

obtained by simple typing from a keyboard and for these a text is written directly. If no then one can use a number of symbol in a font as a special character. For example, the character (\50) means "(" while (\51) ")". In general, the number after backslash symbol is three digits octal number (zeros before can be omitted). The table of correspondence between symbol and number may be obtained directly by means of postscript program of file "font-sym.ps" for Symbol font which may be obtained from the author of this document. It is easy to change the file for other fonts. There are special characters:

\n - newline
\r - return character (ASCII 13)
\t - tab character (ASCII 9)
\b - backspace character (ASCII 8)
\\ - backslash character (ASCII 92)
\(- "(" \50 (ASCII 40)
\) - ")" \51 (ASCII 41)
\[end-of-line] - backslash character before the end of line is used to eliminate the influence of [end-of-line] character on a work of program, normally [end-of-line] means the blank sign, the combination kills this propertiy.

A definition of a symbol by its number is very useful for a Symbol font which contains Greek letters and other special signs. For example, [14 /Symbol f 80 40 m (\150\ \50\155\51) s].

A rotation on 10 degree makes italic Symbol font from normal Symbol font. However, this operation must be applied to each symbol separately. For example, [/sr{-10 r s 10 r}bd 14 /Symbol f 50 60 m (\150)sr 50 70 m (\155)sr] and so on.

For a text in line one may combine different fonts directly with a use of short definitions of head file as follows: [25 35 m (\150\ \50\155)G (m\51)L]. In this case a cursor position moves automatically after each letter.

A3.8 Other possibilities

There are many other operators in PS language for making some more complicated work. For example, the head file defines the command

/sc {dup strigwidth pop -0.5 mul 0 rm s} bd

to show the text with the middle at the current position with 1 argument - the text itself in round brackets. The command uses three new operators: **dup**, **stringwidth** and **pop**. The operator (dup) makes a duplicate of the argument in a stack, the operator (stringwidth) uses one copy of argument from a stack - text - and return in a stack the value - width of the text, the operator (pop) just takes this value from a stack for a use in (mul) and (rm). The operator (s) takes the second copy of text. One can understand how to place the text by its end on the current position or any relative points of the text string.

There are other possibilities to work with the texts and symbols as well as to create the user defined vector fonts as it was done in the file 'cyr30fnt.psf'. The matrix font can be created also. The text string may be allocated along the different paths and the letters may be filled by different ways.

Another possibilities are clipping the images and creating the bitmap images both black-white and color. In reality it is useful to make the postscript fragment of the bitmap image by means of some graphical system like a scaner (for photos) or picture convertors and then used it in VKPS program as a direct PS fragment.

Nevertheless the hexadecimal description of the bitmap image is rather simple. Here the operators are presented which were used for creating the portrait on the title page of this document.

```
/picstr 200 string def
/psppic {gsave 200 266 8
[200 0 0 266 0 -266]
{currentfile picstr readhexstring pop}
image grestore } def
0 -266 translate 200 266 scale
psppic
F4F1F3F4F5F5F1F1F4F3F0F4F7F6F5F4F7F1F0F4F6
F0F4F1F6F4F5F4F1F4F6F4F4F1F3F4F5F5F1F1F4F3
and so on
```

Let us discuss what mean the above written operators. The portrait was obtained by 'Paint Shop Pro' (psp) program. This program first defines the procedure (psppic) and then uses it. The (psppic) procedure uses the string (picstr) which was defined previously as 200 bytes length. The main operator in the procedure is (image). It has many arguments which go just after the operator (gsave). First 3 numbers: 200 266 8 declare that the picture has 200*266 pixels with 8 bits (1 byte) per pixel. Therefore the size of the hexadecimal picture will be 200*266*2 = 106400 bytes because each byte is described by two-byte hexadecimal number. The matrix [200 0 0 266 0 -266] declare the way of

reading the picture. First three numbers for width and next three numbers for height. The structure of matrix just declare to read from left to right and from bottom to top. The next argument is the procedure how to get the data of pixels. It contains the operator (readhexstring) with two arguments - the file and the string, the argument (currentfile) means reading from the same file as the command, the argument (picstr) is the name of string which obtains the data. The operator (pop) is considered above.

Before a usage of the procedure the environment operators (translate) and scale declare the left-bottom corner of the image at (0,0) pt and the size of pixel as 1 pt. Later the image may be retranslate and rescale. The image itself is placed just after the procedure (psppic) pixel by pixel from left-top corner. The size of line is not important. However all 200*266=53200 bytes must be presented.

The more standard system is to use matrix [200 0 0 -266 0 266] and 0 0 translate. In this case the data are read from top to bottom. The matrix may be defined in such a way that the scale operator will have 1 1 arguments. The picture also may be presented without a procedure and there are some more peculiarities. The above program describes the black-white image. To describe the color image the program must be only slightly changed, namely, the operator (image) must be replaced by (false 3 colorimage) which means RGB color representation and each pixel must be described by three bytes - one for read, one for green and one for blue.

The coordinate transformation matrix (CTM) is a rather useful object of PS language. The (translate), (scale), (rotate) and other environment operator are the particular cases of the general CTM. The CTM is defined as CTM = [a b c d tx ty] where the transformation is defined as x' = a*x + c*y + tx; y' = b*x + d*y +ty. For example, (X Y translate) is equivalent CTM = [1 0 0 1 X Y]; (X Y scale) is equivalent CTM = [X 0 0 Y 0 0]; (A rotate) is equivalent CTM = [cos(a) sin(a) -sin(a) cos(a) 0 0] where a=3.14159*A/180. Different CTM matrixes are multiplied by definite procedure.