Dutils unit

Utilities for Delphi

Overview

Dutils unit includes more than 60 procedures and functions that can be used from the Delphi code. The actual platform is Windows 10 and RAD Studio 10.2, but most of the procedures can be used also with other versions of Delphi and Windows. *Dutils* contains tools that may help you to handle screen-related code, creation of custom menus, confirmation and information boxes, 2d and 3d function plots, system of linear equations, matrix operations, other useful mathematic items, binary sorting, spline functions and more.

You obtain the unit in the form of .pas file, more than 1900 lines. You can use or edit the code as you wish, including the whole unit or a part of it to your source code.

To use Dutils, put the "dutils" item inside your uses clause, and make the unit visible from your code (paste it into the same directory or update the corresponding search path in Project->Options->Directories.

It is supposed that you use Dutils within a Delphi form application project. Suppose also that the main form of your project is Form1.

Dutils requaires the following events of Form1 to be defined (in Form->Events):

```
{but, xmsz, ymsz, maxx, maxy and xf0 are declared in Dutils}

procedure TForm1.FormClick(Sender: TObject);
begin
but:=1; form1.Tag:=1
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
xf0:=key
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
if button=mbleft then begin but:=1; form1.tag:=2 end;
if button=mbright then begin but:=2; form1.tag:=3 end;
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
xmsz:=x; ymsz:=y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
form1.tag:=0; but:=0;
end;

procedure TForm1.FormResize(Sender: TObject);
begin
form1.tag:=2; but:=3;
```

```
maxx:=form1.clientwidth; maxy:=form1.clientheight;
end;
```

The meaning of the predefined variables is as follows.
but - auxitiary integer variable
xf0 - auxiliary char variable
xmsz, ymsz - actual cursor position in pixels, respectively
maxx - client width of form1
maxy - client height of form1

The Dutils uses clause:

```
uses Windows,Messages,stdctrls,Classes,Graphics,Controls,Forms,
Dialogs,dateutils,ValEdit,Menus,extctrls,SysUtils,ComCtrls,dateutils;
```

Check if all the items in uses clause are available from the environment of your project.

In Dutils some types and variables are defined in the interfase section, so do not use their names for other purpose. The Dutils declarations are as follows.

```
type tmatrx=array[1..3,1..3] of real;
tprawe=array[1..3] of real;
tabla1=ARRAY[1..500] OF real;
tabla2=array[1..10,1..500] of real;
textar=array[1..50] of string;
t50=array[1..50] of string;
mttp=array[1..100,1..100] of single; {for powplb}
txxp=ARRAY[1..4,1..3] OF real;
uszko=RECORD{point 3D for powplb}
 z:txxp; vx,vy,vz,xcc,ycc,zcc,odl,csn,h:real
end;
PUUp=ARRAY[1..10000] OF uszko;
pup=^puup; {for powplb}
tdisab=array[1..50] of boolean; {for wingr50}
tpop=array[1..20] of textar; {for wingrpop}
HIST=ARRAY[1..2,1..100] OF REAL;
tabspline=array[1..5000] of real;{for splinf}
vecpr=array[1..30] of real;{for ELGAUS, splin}
matvec=array[1..900] of real;{for ELGAUS}

var but,but2:integer;
xmsz,ymsz,maxx,maxy:integer;
zapxy:tbitmap; {for zapscr2}
cnv:tcanvas; {Actual canvas for drawings}
frm:tform;{frm - actual form}
xf0:char;
detera:real;{for deter}
loutyy,gdziey:integer;{Actual line for outlne, define as zero at the beginning}
{gdziey return the actual vertical position for outlne, in pixels}
cgr1,cgr2,cgr3,csil1,csil2,csil3:byte;
powplalfa,powplbeta:real;{angles for powplb}
```

In the following sections you can find the descriptions of the Dutils procedures and functions.

Screen-related tools

Form dimensions
procedure dajwymiar(y:tform);
Defines the form dimensions of the form y, stores the form client width and height in mzxx and maxy, respectively.

Save screen image
procedure zapscr;
Stores the screen image in file *obrazek*.

Retrieve screen image
procedure czyts;
Reads the screen image stored by zapscr;

Clear rectangle
procedure czysrec(x:tcanvas; x0,y0,x1,y1:integer; a,b,c:byte);
Fills the rectangle (x0,y0,x1,y1) of canvas x with color rgb (a,b,c)

Store screen2
procedure zapscr2;
Stores the form canvas image in bitmap *zapxy*. The bitmap is created automatically in the initialization of Dutils.

Retrieve screen2
procedure czyts2;
Reads the canvas image sored by *zapscr2*.

Store rectangular image
procedure zaprect(a,b,c,s:integer);
Stores a rectangular region image in file obr.dat.
Top left vertice at (a,b), bottom right in (c,d);

Retrieve rectangular image
procedure czytrect(a,b:integer);
Reads rectangle image stored by zaprect. The upper left corner of the image is located at (a,b).

Move cursor
procedure movcur(x,y:integer);
Moves the cursor to (x,y);

Clear screen with color
procedure czys(x:tcanvas;a,b,c:byte);
Fills the canvas x with color rgb:a,b,c.

Set pen color
procedure sc(x:tcanvas;r,g,b:byte);

Set the pen color (r,g,b) for canvas x.

Set brush color

procedure filc(x:tcanvas;r,g,b:byte);
Set the brush color (r,g,b) for canvas x.

Display a text

procedure out0(x:tcanvas;a,b:integer;cl:tcolor;t:string);
Displays the string t on the canvas x, at (a,b) with text color cl.

Display consecutive text line

procedure outlne(x:tcanvas;cl:tcolor;t:string);
Each call to outlne displays consecutive text line on canvas.
The line number grows automatically, store in the global variable loutyy
If the first char "|" reight allign
if the first char "%" does not wait at the end of the page

Wait for mouse

procedure zaczek;
Waits for a mouse click.

Display a message

procedure pisak(x:tcanvas;a,b:integer; c1,c2,c3:byte; t1,t2,t3,t4:string);
Displays up to four lines of text (t1,t2,t3,t4) on the canvas x, The background color is given by rgb c1,c2,c3.

Draw a line

procedure line(x:tcanvas;a,b,c,d:integer);
Draws a line on canvas x using actual pen color.

Draws a transparent rectangle

procedure czworob(xc:tcanvas;a,b,c,d:integer);
Draws a rectangle on the canvas xc, using actual pen color. Does not fill the rectangle.

3D poin-to-screen coordinates

procedure ekrancr(x,y,z,dlx,al,be:real; var xs,ys:real);
Returns in xs,ys the 2d screen coordinates of the 3d point (x,y,z).
dlx - distance to eye
al,bet - view angles
It is supposed that the point is with a cube, where x,y,z belong to [-1,1]
Uses the auxillianction *xyzz.*

Binary sorting

Dutils includes tools for fast sorting, using a binary tree. Sorting 10000 real numbers takes only 10 milliseconds, and sorting one million elements needs about two seconds.

Creating binary tree

procedure makebin(var n:integer);

This creates a binary tree that can be used to sort records. The key to sort is a real number. The sorted records are defined by the following.

```
type tbintr=record
  {element of the binary tree:}
  i,id,w:integer;  vx:real; {vx - key value for sorting}
  {i - index in array artree, id - original index (may be moved with vx), w - level}
  fir,sec,sup:pointer {pointers to the successors fir and sec, and to the
    superior (parent) element}
   {.....You can add here other components, if necessary.....}
end;
```

variable *vx* is the key. The records are sorted in ascending order. See procedure *sortbin* below for an example of application.

Insert an element to the tree

procedure insr(k:integer; x:real);

Each call to this procedure inserts a new element x to the tree. The element is a real number x, which is treated as the sorting key. It is also stored in the array *artree* as the value of the component vx of the record *tbintr.* The component id fo *tbintr* is set equal to k.

Displaying the tree

procedure showbin;

This procedure shows the tree created by makebin. It work for small trees, up to 100 elements.

Retrieving the consecutive minimal elements

function dajmin:tbintr;

A call to dajmin returns a consecutive record with minimal value of vx. . See procedure *sortbin* below for an example of application.

Sorting example

procedure sortbin(n:integer);

This procedure creates a binary tree with n (up to 1000000) elements. The key values are generated as random numbers between 0 and 1. The tree is created in such way that the uppermost (top) element has the minimal value of the key vx. Then, the consecutive minimal value records are retrieved from the tree, stored and displayed. Sorting time for n=1000000 is approximately two seconds, and for n=10000 it takes 10 seconds of computer time.

Menus and dialogs

The use of Dutils menus is different from the Delphi main menu of the form. The Dutils menus are created dynamically by calls from the user code. The menu return the number of the selected option. Then the user can code anything he wants directly in the same code segment using the *case* or *if* instructions.

Simple menu

```
procedure wingrpanel(frm:tform; a,b:integer; nag:string; clo:tcolor;
var t:textar; disab:tdisab; lmin:integer; var opt:integer);
```

Creates a simple, horizontal menu.
frm - actual form
a,b, - position on the screen
nag - caption
clo - background color
t - option strings
disab - boolean array. If disab[n] true, option n disabled
lmin - minimal option width
returns opt - selected option

t is an array of 50 strings. However, the number of strings (options) is limited by the form dimensions. If you define *n* strings, the string *n+1* must be empty.
The selected option is returned in variable opt.
Example of use: menu with six options, minimal width of option box equal to 20.

```
var te:t50;  k:integer
begin
for k:=1 to 50 do disab[k]:=false;
t[1]:='First option'; t[2]:='Second';
t[3]:='Other'; t[4]:='Another'; t[5]:='Fifth';
t[6]:='Quit'; t[7]:='';
wingrpanel(form1, 30,30,'Mission possible',clgreen,t,disab,20,opt);
......
......
end;
```

Horizontal menu

procedure wingrtop(frm:tform; clo:tcolor; t:textar; disab:tdisab; var opt:integer);
Horizontal menu at the top of the form. The use is similar to wingrpanel. Remember that in the procedure form1.formclick must be but:=1.

Popup menus

procedure wingrpop(frm:tform; cl,cl2:tcolor; t:textar; t2:tpop;
disab:tdisab; var opt,opt2:integer);
Upper menu with popups
t - captions of the main menu
t2[opt] array of captions for submenu
returns opt - options of the main menu, opt2 - submenu option

Note that t2 is a two-demensional array of strings. Before tha call to wingrpop you must define strings for the main menu in the arra t, and the strings for sub-menus. The string in t2[5][3] is the caption of the third option of the submenu invoked by option 5 of the main menu. For example, if the number of options of menu 5 is equal to 7, then t2[5][8] must be an empty string. As in the wingrtop, if the number of option of the main menu is equal to 6, the string t[7] must be empty. If, for example, the item 4 of the main menu has no corresponding popup, then t2[4][1] must be empty.

If an option of a submenu has caption *Cancel, cancel* or *CANCE*L, the the control goes back to the main menu. If no such option is defined, the cancel option is added automatically.
As the result,, the option of the main menu is returned in *opt*, and the selected option of the popup is in *opt2*.

Confirmation dialog

function confirm(frm :Tform; a,b:integer; nag:string):boolean;
This displays a short message or question defined in *nag*, and two buttons with captions YES and NO. The program waits for user click on one of these buttons and returns *true* or *false* respectively. The whole box can be moved with mouse drag.
frm - parent form
a,b - position, nag - caption

Information box

procedure inform(frm :Tform; a,b:integer; nag1,nag2,nag3,nag4:string);
Displays up to four lines of text (nag1,nag2,nag3,nag4) on the canvas x. Waits for the click on the *Accep*t button. *frm* is the actual form.

Read several items

procedure czyt50(frm:tform; ile,x,y:integer; t:t50; nag:string; var v:t50);
This can be used to enter up to 50 strings. frm is the actual form, ile is number of items to enter, (x,y) is the position of the upper left corner of the main box, t is the array of item captions (what to enter), and v is the array of strings with the results.

Plots and other graphics

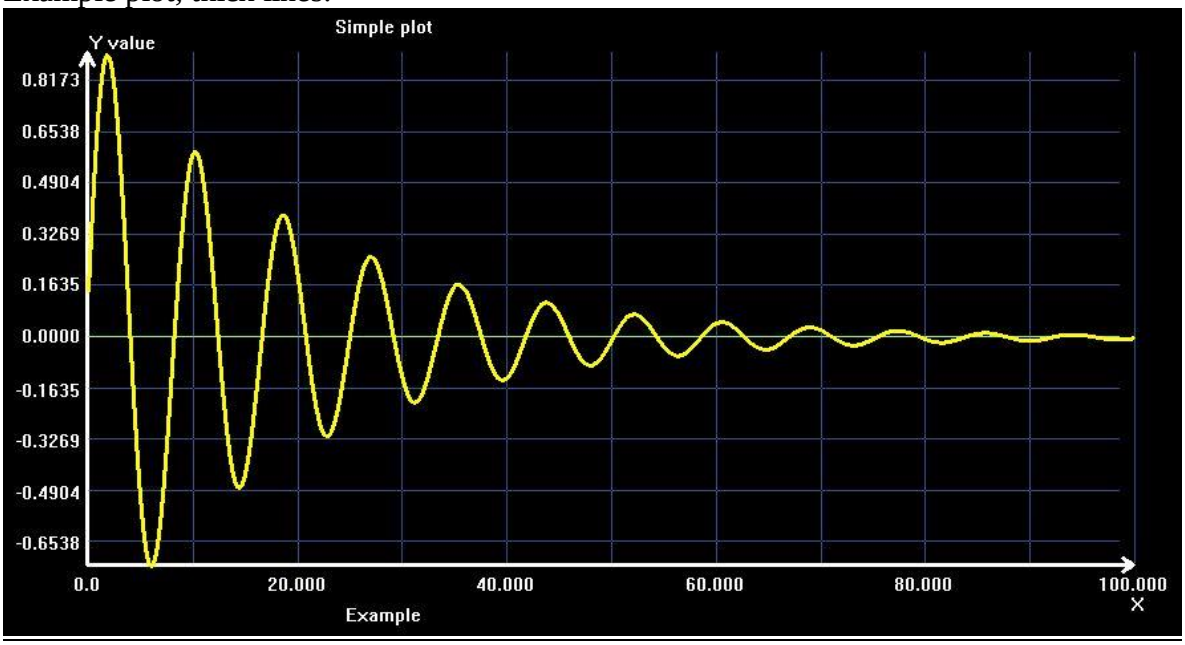
Function plot

procedure trzd(xc:tcanvas; w1,w2,w3,w4,n:integer; var z:tabla1;
xma,xmi:real;reg,clear,col,krata:boolean; dol,gora:real; jak:integer;
t1,t2,osx,osy:string;ch:integer;low,upp:real;colo:tcolor);

Trzd displays the plot of a function, on canvas xc.
Parameters:
xc - canvas to plot
w1,w2,w3,w4 - the plot area
z - function values to plot, up to 500 points
n - number of points in z, <=500
xma,xmi - limits min and max if not normalizes
reg=true to normalize
clear=true to clear area before and after plotting
col=true for black background
krata=true to show grid
dol,gora - min and max value for x axis
jak=1 plot with thick line, 0 for normal
t1,t2 - upper and lower captions
osx, osy - X and Y axis captions
ch if >0 then puts a ch number on the curve

if ch<0 then does not displays comment "variable does not change"
colo - line color if not B/W plot

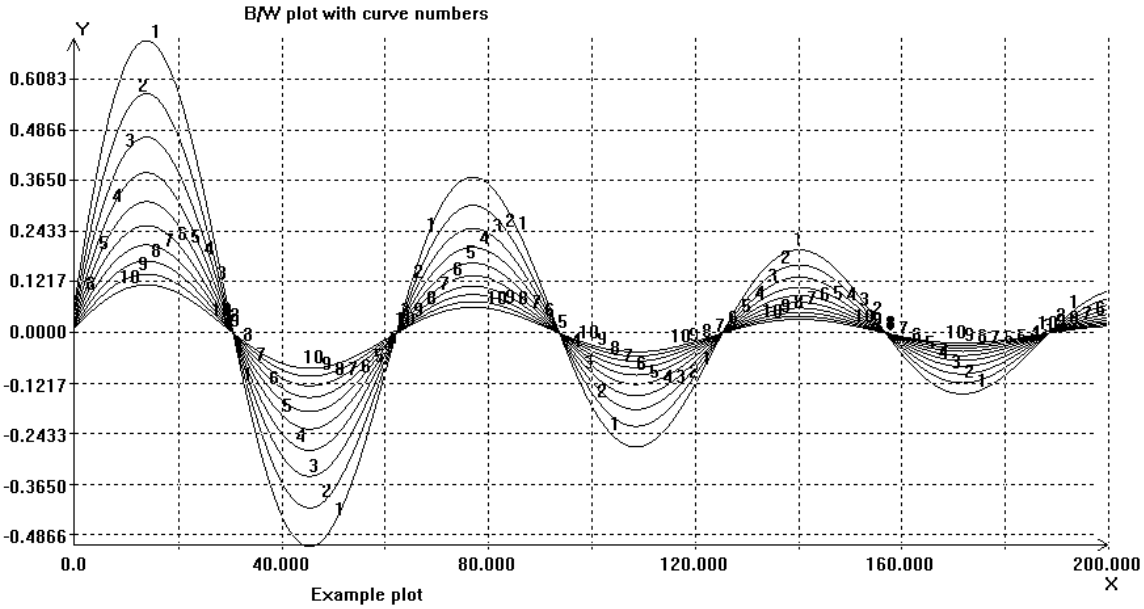
Example plot, thick lines:



Multiple plot

procedure multiplo(xc:tcanvas;n,nkur:integer;
w1,w2,w3,w4:integer;varxz:tabla2;xma,xmi:real; reg,clear,col,krata:boolean; dol,gora:real;
jak,tak:integer; t1,t2,osx,osy:string);
Multiple plot, up to 10 functions.
xc - canvas to plot
xz - data to plot, xz[curve number,point number
up to 10 curves, 500 points per curve
if tak>0 puts curve numbers on the plot
Other parameters as in *trzd*

Example:

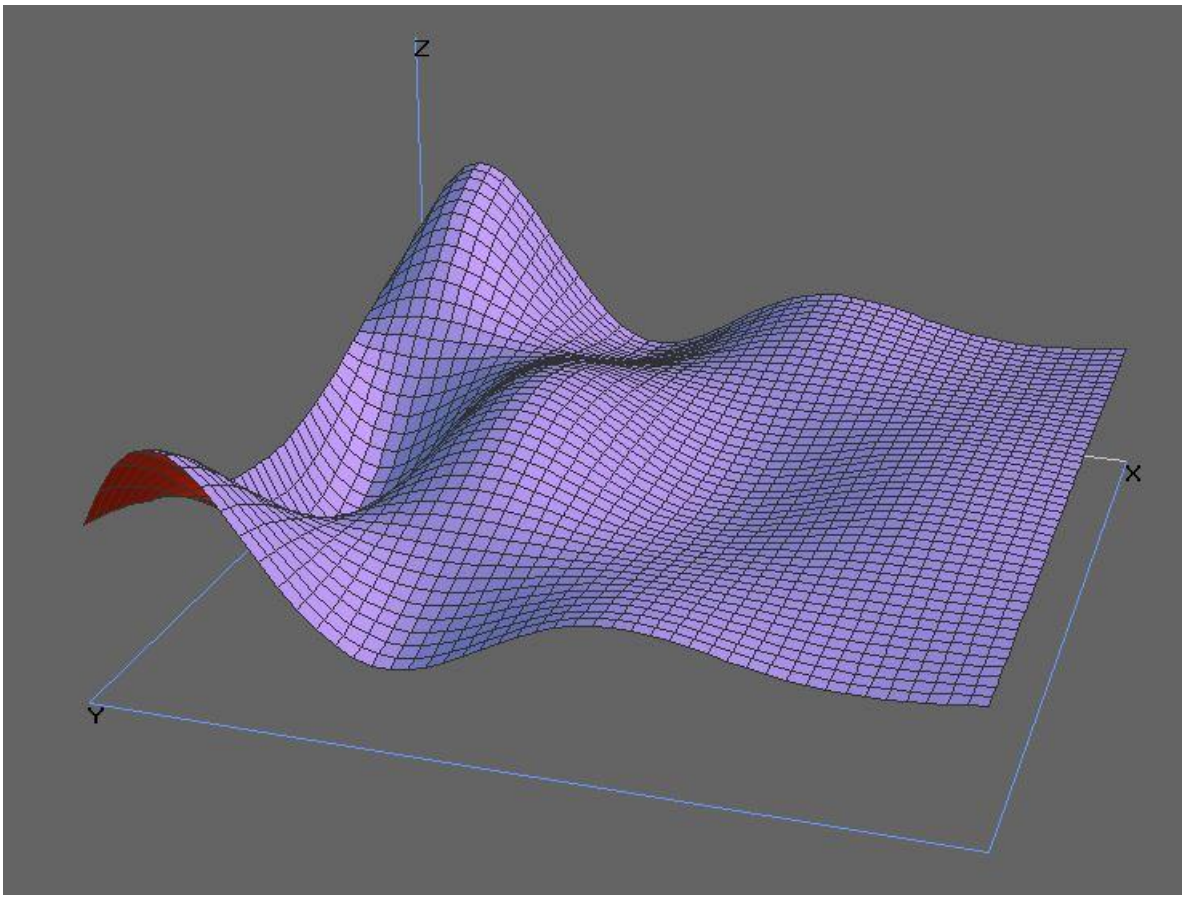


3D plot of a function of two variables

procedure powplb(xc:tcanvas;n,m:integer;alf,bet:real;var x:mttp;jak,
tak:integer;osx,osy,osz:string; bc1,bc2,bc3:byte);

powplb displays the 3d image of the plot of a function of two variables.
The values to plot must be defined in the 100x100 matrix x of type mttp
xc - canvas to plot, alf,bet - view angles osx,osy,osz - X,Y and Z captions resp.
JAK = 0 - smooth surface, JAK = 1 lines X direction, JAK = 2 - grid
TAK = 0 color, TAK = 1 black/white
n - X size, m - Y size (grid steps)
n,m - takes abs value
if n<0 then draws and exits, n>0 waits
if m<0 then vertical scale 0.75
if the first character of osz is "&" then does not normalize
jezeli w osz pierwszy znak "&" to nie normalizuje.
Values to plot shoud be between -1.0 and 1.0
in powplalfa and powplbeta returns view angles
bc1,bc2,bc3 - rgb of the background

See the section "Some example application codes" for an example of application.

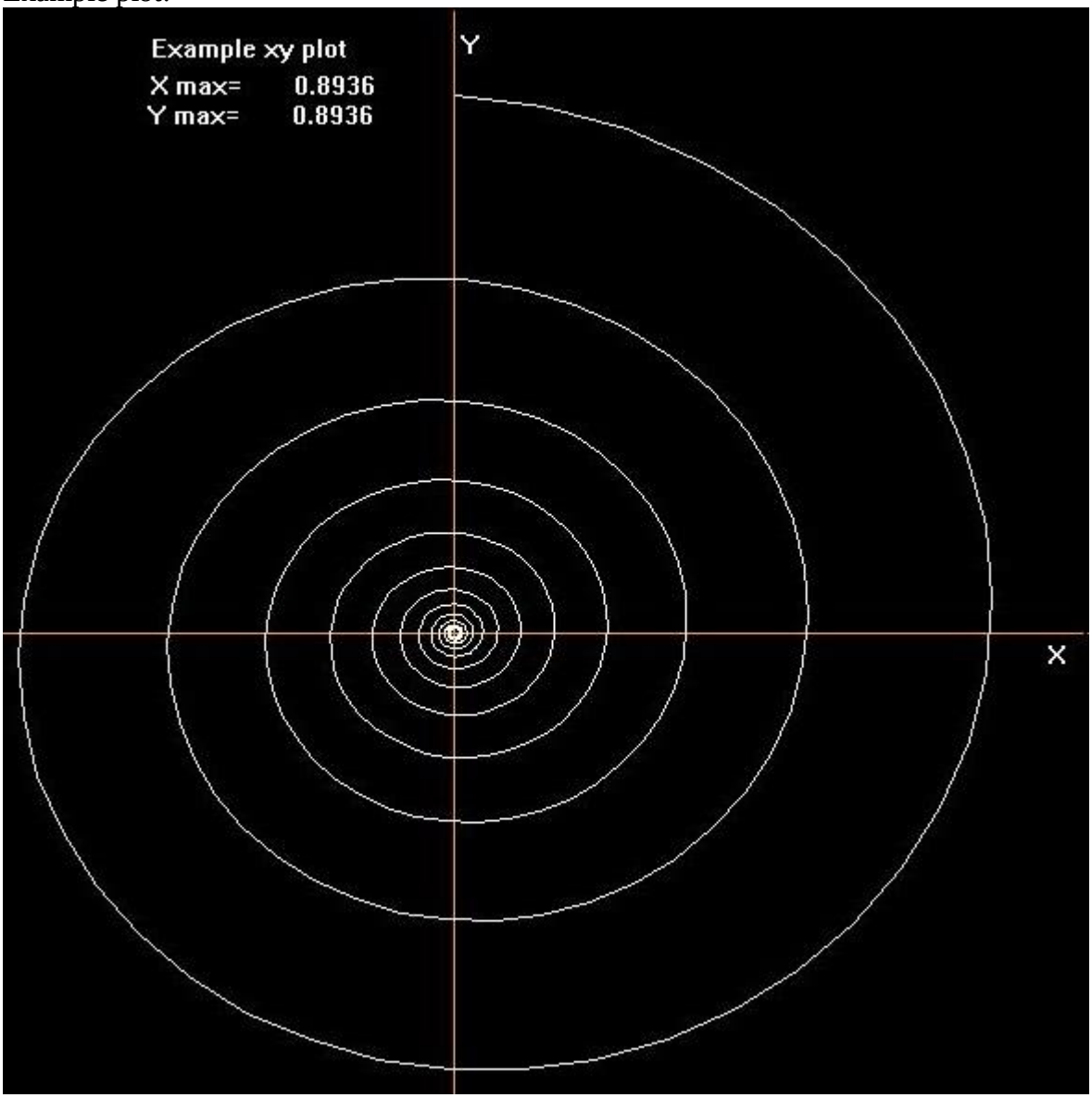


Example: procedure *powplb*

XY plot

procedure xyplot(cnv:tcanvas; k:integer;var x,y:tabla1; xm,ym,wsp:real;
hor,ver,tit:string;ofx,ofy:integer; norm,col:boolean);
{XY plot.
cnv - canvas to plot
k - number of points, takes abs(k)
x,y - data to plot
xm,ym - scale (max)
hor - horizontal axis caption, ver - vertical, tit - title of the plot
ofx,ofy - screen offset x,y in pixels
norm=true to normalize
col=true for black background

Example plot:



Some useful math functions

Linear equations

procedure elgaus(n:integer;var x,pra:vecpr;var ax:matvec;var
err:boolean);

Resolves a set of n linear equations
x - unknown, returns the solution
n - number of equations, max 60
ax - matrix of coefficients as 1-dime array:
ax(kj) -> ax[(k-1)*n+j], k - row, j - column
pra - right-hand sides

err - set true if error

Note that the coefficients of the equations must be stored in one-dimensional arra ax, crow by row. For example, if your left-hand side matrix ix y[k,j] then you shoud store it as follows (k - row, j- column):

```
for k:=1 to n do for j:=1 to n do xc[[(k-1)*n+j] := y[k,j];
```

The right-hand values are provided in one-dimensional array *pra*.
The matrix and right-hand side type are *matvec* and *vecp*r, respectively. If you want to solve greater sets of equations, you can edit the corresponding declarations in *dutils.*
The result is returned in the vector *x:vecpr*.

Point-to-line distance

function pointtoline(a,b,c,x1,y1,z1,x2,y2,z2:real):real;

Returns the distance between a 3d point (a,b,c) and a line defined be two points (x1,y1,z1) and (x2,y2,z2).

Point-to-section distance

function pointtosection(a,b,x1,y1,x2,y2:real):real;
Returns the distance between a 2d point (a,b) and a line section with end points (x1,y1) and (x2,y2).

Point-to-plane distance

function dpunktpow(a,b,c,d,x,y,z:real):real;
Returns the distance from a 3d point (x,y,z) to the plane defined by the equation ax+by+cz+d=0

Plane-triangle intersection

function triangleplane(a,b,c,d,x1,y1,z1,x2,y2,z2,x3,y3,z3:real;
  var ux,uy,uz,vx,vy,vz:real):boolean;

Returns true if a triangle intersects with a plane ax+by+cz+d=0
In ux,uy,uz,vx,vy,vz returns the common line section.
The triange vertices are (x1,y1,z1), (x2,y2,z2) and (x3,y3,z3).
The intersecting section is returned as two points (ux,uy,uz) and (vx,vy,vz)

Determinant 3d

function deter(var a:tmatrx):real;
Returns the value of the determinant of a 3x3 matrix a.

Matrix product

function matrprod(var a,b:tmatrx):tmatrx;
Returns 3x3 matrix:tmatrix that is the result of multiplying matrix a by matrix b.

Matrix-vector product

function matrvec(var a:tmatrx; b:tprawe):tprawe;
Returns the vector that results from multiplying the 3x3 matrix a by the vector b.
function weibull(shape,loc,scale:real):real;

Returns a value of the random variable with Weibull distribution,
the shape>0 is the *shape parameter* scale > 0 is the *scale parameter* of the distribution.
The value loc is the "locus" added to the value, normally equal to zero-

Lognormal distribution

function lognormal(mu,sigm:real):real;
Returns a value of the random variable with lognormal distribution. Mu and sigm are the mean and standard deviation of the variable's natural logarithm,

Uniform distribution

function plusminus(s,d:real):real;
Returns a value of the random variable with uniform distribytion within the interval(s-d,s+d)-

Normal distribution

function norm(s,d:real): real;
Returns a value of the random variable with normal distribution. s is the expected value and d is the standard deviation.

Histogram processing

procedure dist(m:integer; var h:hist);
This procedure processes the histogram stored in table h:hist. The processed histogram can be used to generate random variables with arbitrary distribution, obtained from the real sysem (timing). See the foolowing section sample for more deatil.

Sample (empirical) distribution

function sample(m:integer; h:hist):real;
The *sample* function makes it possible to introduce a user distribution in the form of a histogram and then to generate random numbers according to this distribution
This function returns a real random value generated due to the distribution defined by the programmer and stored in an array. The array is of type type hist=array[1..2,1..m] of real and contains M observations, each observation being a value and the corresponding number of occurrences of the value. The distribution is discrete. If, for example, you want to generate values due to the distribution of service time *ts*, we must discretize the time. The discretization may be as follows: *0 < ts < 1* corresponds to *ts1=0.5, 1<=ts<2* to *ts2=1.5* etc.
In the program define a two-dimensional array array x of type hist as follows.

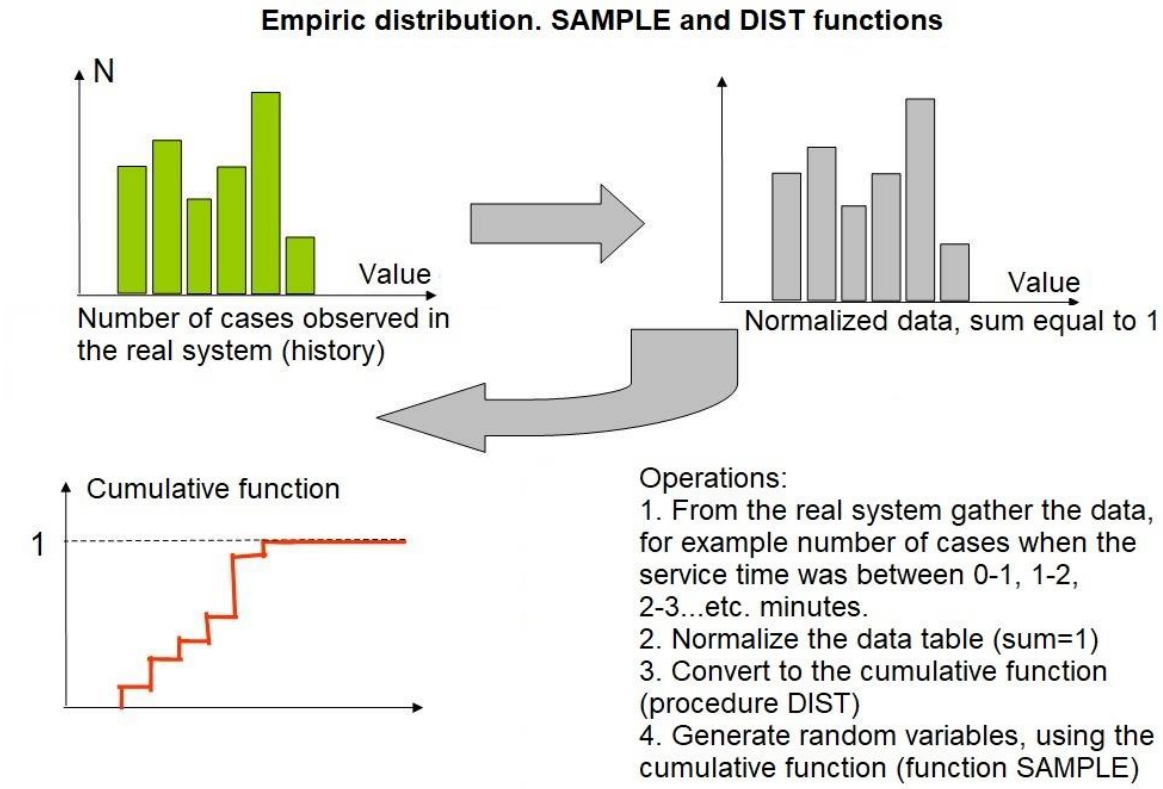
```
x[1,1]:=n1; x[2,1]:=ts1;
x[1,2]:=n2; x[2,2]:=ts2;
- - - - -
- - - - -
x[1,M]:=nM; x[2,M]:=tsM;
```

where ni is the number of clients served with service time *t, tsi- 0.5t<tsi+0.5*. The numbers *ni* should be obtained from observations during sufficiently long time.

Once defined the array , it must be processed by the procedure *dist*(m,x). After this, the array x can be used to generate random service times with the (discretized) distribution of the service model under consideration. The calling expression for the *sample* function is

sample(m,x);

where *m* is the number of quantization intervals and *x* is the array of type *hist*, defined earlier and processed by the procedure *dist*.



Triangular distribution

function triandis(a,b,c:real):real;
Generates a random value with triangular distribution within a<b<c.

Erlang distribution

function erlang(k:integer; s:real):real;
Generates a random value with Erlangr distribution.
k - function order
s - expected value

Exponential distribution

function negexp(s:real):real;
Generates a random value with negative-exponential distribution.
s - expected value
Useful for between-arrival intervals for the Poisson arrival process.

Uniform distribution, discrete and real

function irnd(k:integer):integer;

function rnd:real;
irnd returns random integer value between 0 and k-1
rnd returns random value within (0,1)

Poisson distribution

function poisson(n:integer):integer;
Returns the integer value with the Poisson distribution. n is the parameter of the distribution

Gamma distribution

function gamma(lambda,loc,shape:real):real;
Generates a random value with distribution gamma.
Lambda and shape - gamma distribution parameters, loc - additional value (locus)

Beta distribution

function beta(m,n:real):real;
Generates a random value with distribution beta. m and n are the two parameters of the distribution.

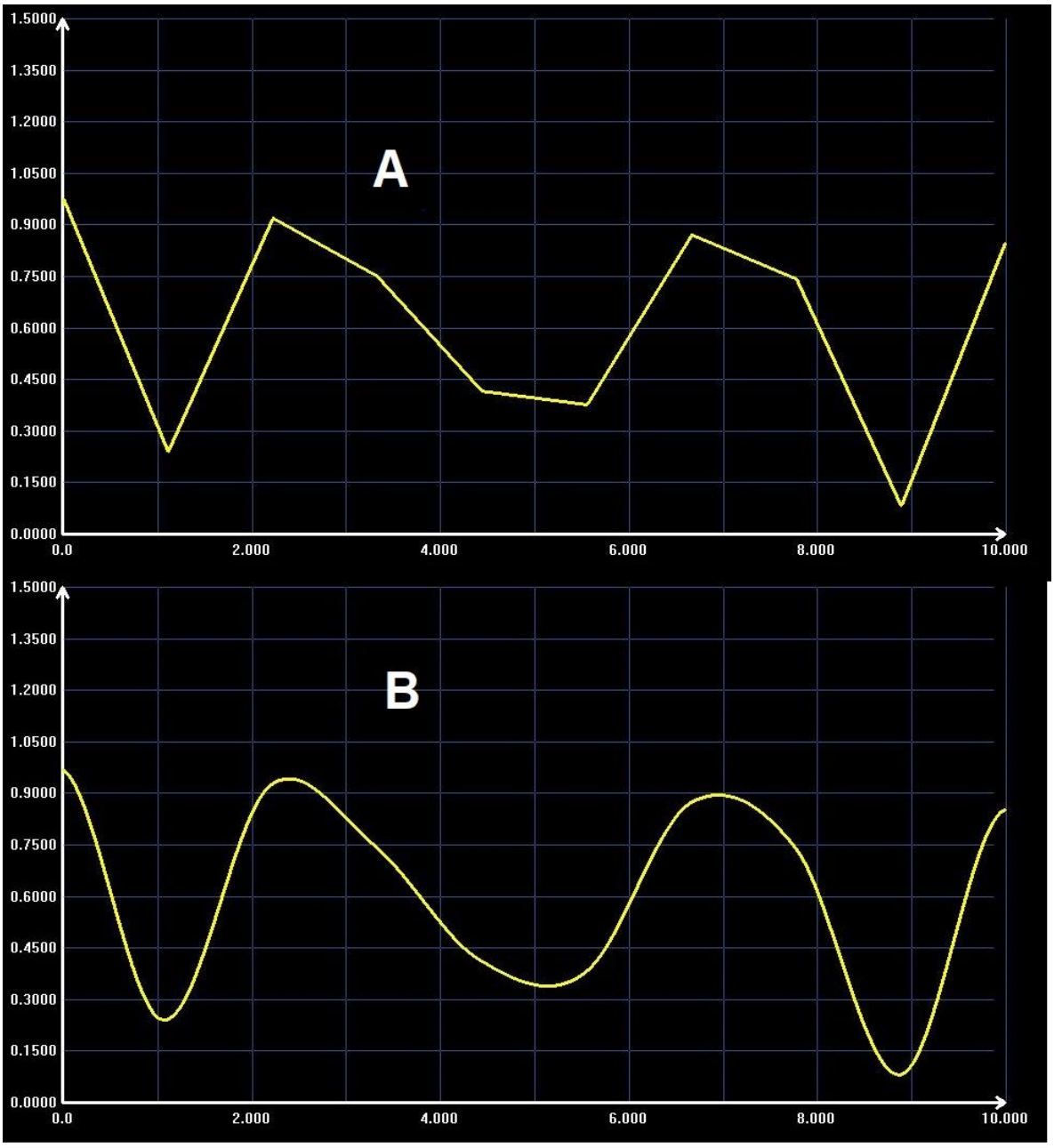
a to power b

function adox(a,x:extended):extended;

Spline functions

procedure splinef(k,j:integer; var x:tabla1; var y:tabspline);
procedure resplin(var k:integer; var z:tabla1);

splinef procedure generates spline functions for a given data represented by an array of x,y paires.
Creates spline function s(x)=axxx+bxx+cx+d
k - number of original points, j - number of new interpolated points
in each interval, j*k<=5000
Uses procedure resplin. The result returned in the array y (smoothed data). See the section section "Some example application codes" for examples of application.
See the example call in section **Some Example Calls**. For example, if you have original data that cover the interval 0 to 10 with step equal to 1.0, you have 11 original points 0,1,2...10. In this case k=11. If the number of interpolated points within each interval is equal to 20, then the *splinef* procedure will create (k-1)*20=200 points of the smoothed curve.
Limitation: 6<k<50.



A - original data B - smoothed with spline functions

Miscellaneous

Copy text files

procedure przewala(a,b:string);
Copy strings from the file a (name of the file) to file b.

Uppercase
procedure upp(var t:string);
Converts a string to uppercase.

Get short file name
procedure filshortek(filak:string; var fila:string);
In *fila* returns short name file, without path and without extension.
For example,
outlne(cnv,clBlack,shortfil("c:\\algo\\otro\\filek.dat",0));
prints *filek*.

Terminate program
procedure adios;
Destroy the forms, terminates the application run.

Some example application codes

Menus, dialogs
```
 procedure testmenus;
{Some examples of calls to dutils menu procedures. Note that the menu is
created dynamiclally and disappears after being used}
var k,j:integer;t:textar; disab:tdisab; opt,opt2:integer ; t2:tpop;
te:t50; v:t50;
begin

for k:=1 to 50 do disab[k]:=false; disab[3]:=true; disab[5]:=true;
{Simple menu:}
t[1]:='Something'; t[2]:='Another item';
t[3]:='Third'; t[4]:='Fourth'; t[5]:='';
czys(cnv,0,0,200);
wingrpanel(form1, 30,30,'Mission possible',
claqua,t,disab,20,opt);
czys(cnv,0,0,0);
out0(cnv,30,30,clwhite,'Retrned value: '+inttostr(opt));
zaczek;

{Horizontal menu}
t[1]:='Something'; t[2]:='Another item';
t[3]:='Third'; t[4]:='Fourth'; t[5]:='Type something';
t[6]:='Nothing'; t[7]:='';
czys(cnv,200,200,200);
wingrtop(form1,clblue,t,disab,opt);
czys(cnv,0,0,0);
out0(cnv,30,30,clwhite,'Retrned value: '+inttostr(opt));
zaczek;

{Menu with popups}
for k:=1 to 50 do disab[k]:=false;
t[1]:='First option'; t[2]:='Second';
t[3]:='Other'; t[4]:='Another'; t[5]:='Fifth';
t[6]:='The last one'; t[7]:='';
```

```
for k:=1 to 10 do for j:=1 to 10 do t2[k][j]:='Option '+inttostr(j);
for k:=1 to 10 do t2[k][8]:='';
t2[6][1]:=''; {option 6 without sub-menu}
t2[2][3]:='cancel';
czys(cnv,0,200,0);
wingrpop(form1,clblue,clred, t,t2,disab, opt,opt2);
outlne(frm.Canvas,clwhite,'Returned option values: '+
inttostr(opt)+'  '+inttostr(opt2)); zaczek;

{read 35 strings}
for k:=1 to 25 do begin te[k]:=inttostr(k); v[k]:='String '+inttostr(k) end;
te[3]:='Something'; te[4]:='Else';
te[7]:='Seventh'; v[20]:='What is it?';
v[8]:='????';
czys(cnv,0,0,0);
czyt50(form1,25,200,20,te,'Enter the values', v);
outlne(cnv,clwhite,'Results');
for k:=1 to 25 do outlne(cnv,clwhite,v[k]);

{Confirm dialog}
czys(cnv,0,0,220);
if confirm(form1,200,200,'Yes or not?')
then out0(cnv,400,40,clwhite,'Confirmed: Yes')
else out0(cnv,400,40,clred,'Confirmed: No');
zaczek;
end;
```

Plots

```
procedure plotexamples;
var k,j,jak,tak,i:integer; xv:mttp; x,y:tabla1; xm,xi,ym,yi:real;
xz:tabla2;
begin
{Simple plot}
for k:=1 to 500 do y[k]:=sin(k*0.15)*exp(-0.01*k);
trzd(cnv,100,100,maxx-100,maxy-100,500,y,2.0,0.0,
true,true,true,true,0.0,100.0, 1,'Simple plot','Example','X','Y value',0, 0.0,10.0,
clyellow);
{Multiple plot}
for j:=1 to 10 do for k:=1 to 200 do xz[j,k]:=sin(k*0.1)*exp(-0.01*(k+20*j));
multiplo(cnv,200,10,100,100,maxx-100,maxy-100,xz,1.0,-1.0,true,true,false,true,
0.0,200, 0,1,'B/W plot with curve numbers','Example plot','X','Y');
multiplo(cnv,200,10,100,100,maxx-100,maxy-100,xz,1.0,-1.0,true,true,true,true,
0.0,200, 0,0,'Plot without curve numbers','Example plot','X','Y');
{3d plot}
i:=50;
for k:=1 to i do for j:=1 to i do
  xv[k,j]:=sin(k*0.25)*cos(j*0.12)*20*exp(-0.07*k);
jak:=2; tak:=0;
powplb(cnv,i,-i,0.4,0.3,xv,2,0,'X','Y','Z',0,0,100);
out0(cnv,20,10,clwhite,'Without grid');
powplb(cnv,i,-i,0.3,0.1,xv,0,0,'X','Y','Z without grid',0,0,100);
```

```
{XY plot:}
xm:=1.0; xi:=-1.0; ym:=1.0; yi:=-1.0;
for k:=1 to 500 do begin
 x[k]:=sin((k-1)*0.15)*exp(-0.01*k); y[k]:=cos((k-1)*0.15)*exp(-0.01*k) end;
czys(cnv,0,0,0);
xyplot(cnv,500,x,y,xm,xi,ym,yi, 'X','Y','Example xy plot',300,100,true);
zaczek;
end;
```

Spline functions

```
procedure splins;
var z:tabla1; y:tabspline; k,n:integer;
begin
{Spline functions}
n:=11;
for k:=1 to n do z[k]:=random;
czys(cnv,0,0,0);
trzd(cnv,50,50,maxx-50,maxy-50,n,z,1.5,0.0, false,false,true,true,
0,10,1,'','Original data','','',0,0,1, clyellow);
zaczek;  czys(cnv,0,0,0);
splinef(n,20,z,y);
for k:=1 to (n-1)*20 do z[k]:=y[k];
trzd(cnv,50,50,maxx-50,maxy-50,(n-1)*20,z,1.5,0.0, false,false,true,true,
0,10,1,'','Splie curves','','',0,0,1, clyellow);
zaczek;
end;
```