



In the name of Allah,
Most Gracious, Most Merciful.



Article Title :

Miniatoric Infinity Plot And It,s Applications



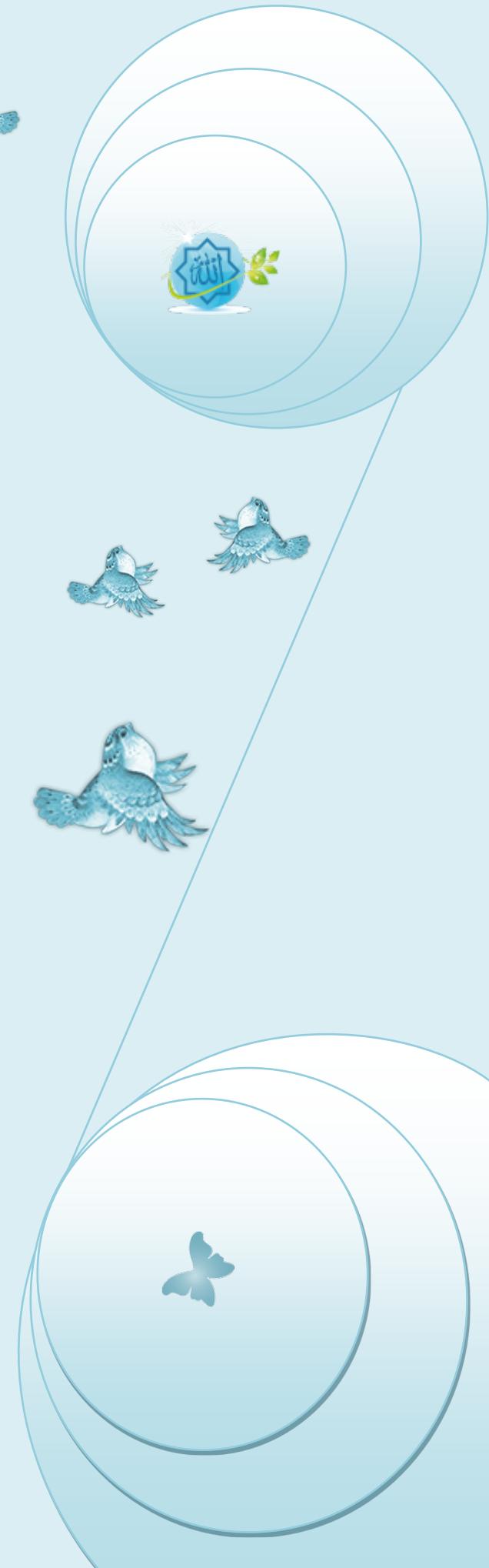
Digest Description Of Miniatoric Infinity Plot Concept :

- If either of the range end points of the horizontal range contains $\pm\infty$, an infinity plot is generated.
- An infinity plot is obtained by transforming $-\infty$.. $+\infty$ to $-\frac{\pi}{2} \dots \frac{\pi}{2}$ by a transformation that approximates arctan. This is a nice way of getting the entire picture of $f(x)$ on the display.
- Such a graph, although distorted near $x = -\infty$ and $+\infty$, contains a lot of information about the features of $f(x)$.



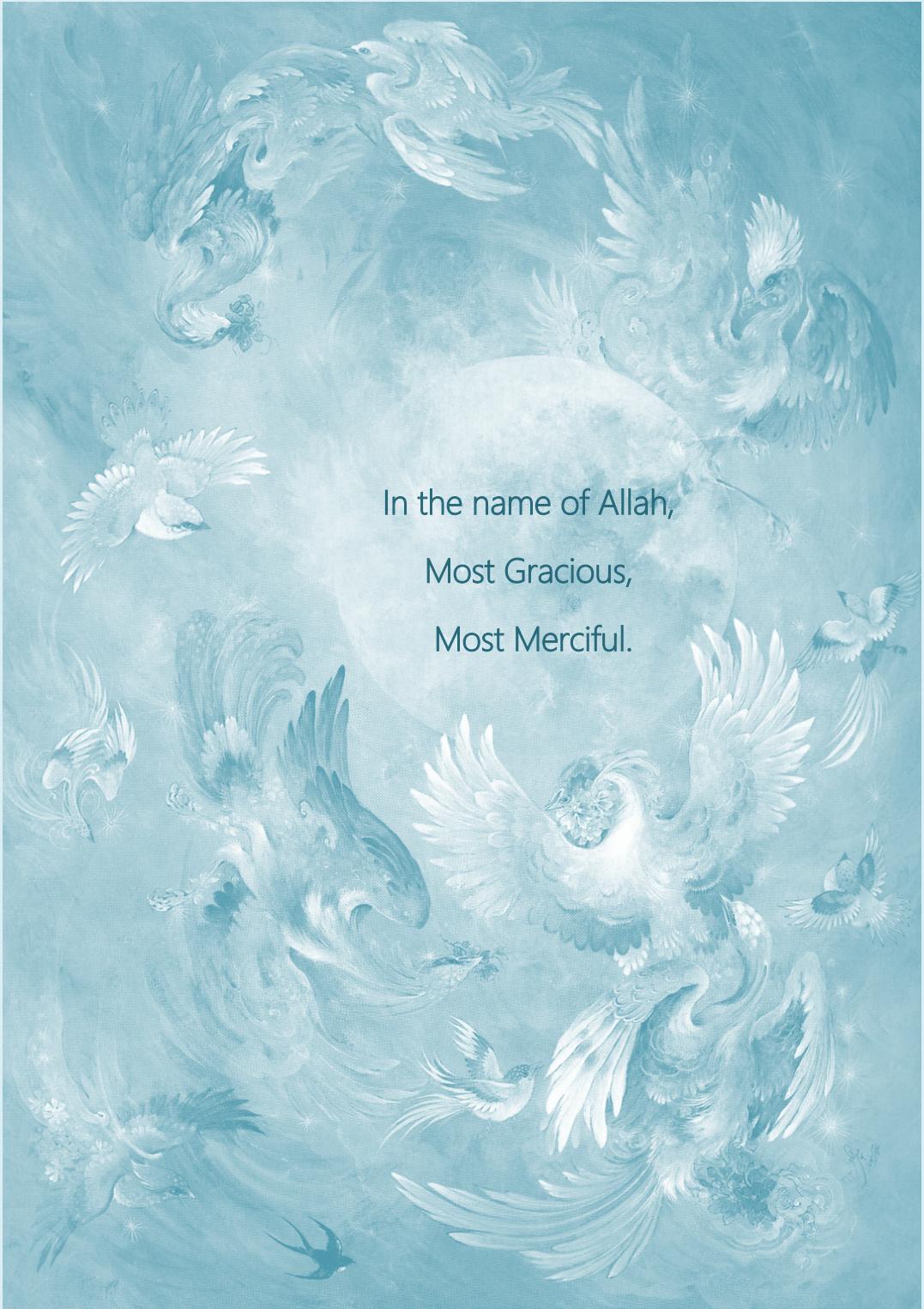
The Article Is A Gift
From The GOD
To All The Believers In GOD

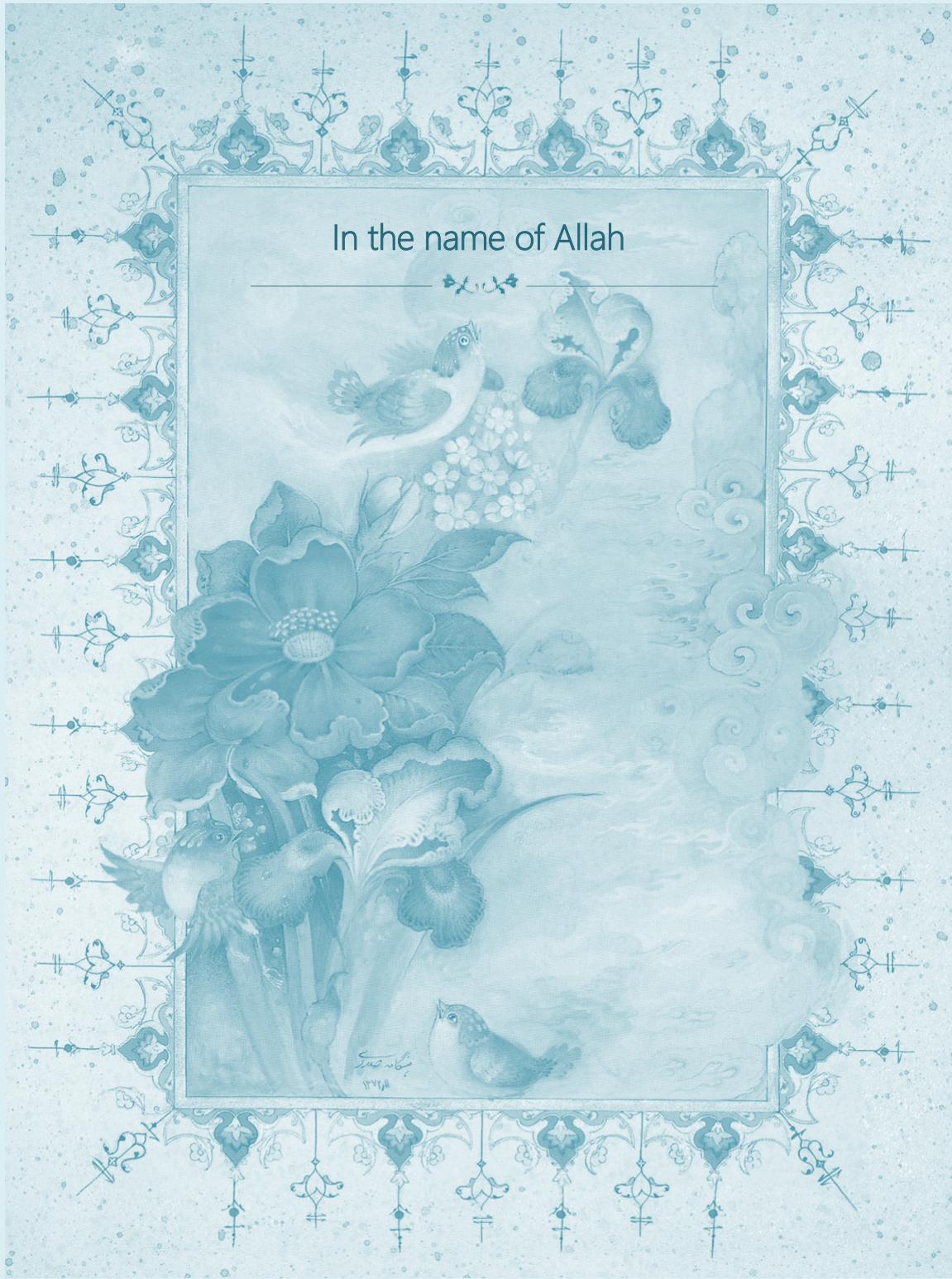
Do Not Forget The Ordained Prayer To GOD
Because Of This Great Manifest Signs Of The GOD.





In the name of Allah,
Most Gracious,
Most Merciful.







In the name of Allah





In the name of Allah





INFINITY PLOT APPLICATIONS

- **BrentGhofrani Method to find roots**

The function **BrentGhofrani** computes the root of a function within a given interval via the Brent Ghofrani method. This function combines the guaranteed convergence of the Bisection method and the speed of the Newton-Raphson method (which, however, may diverge). The **Brent Ghofrani** method is always converging, and it is usually much faster than Bisection method. Due to these advantages, it is considered as the method of choice for root finding. For convenience, the function **BrentGhofrani** has the same syntax as **Bisect**.

Syntax

```
Root = BrentGhofrani(f,xl,xr,eps,maxit,show)
```

Returns the roots of the function **f** inside the interval **{xl,xr}**. The arguments **eps**, **maxit** and **show** are optional. **eps** defines the desired accuracy (default: **Epsilon**, i.e., 1.12×10^{-16}); it can be set to "auto", which is equivalent to the default accuracy. **maxit** defines the maximum number of iterations (default: 100). **show** is a boolean argument that controls whether progress of the iterative process will be displayed or not (default: **false**); if set to **true**, the function value, **f(root)**, at each iteration will be displayed.

Example

The example program **XBrentGh** uses the function **BrentGhofrani** to find the root of the function $e^{1-x} - \frac{x}{2}$ within the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. In order to check the result, the program computes the function value at the root obtained, and shows the result graphically.

The Figure shows the results obtained by running this program. Note that function value **f(root)** is too close to **zero**.

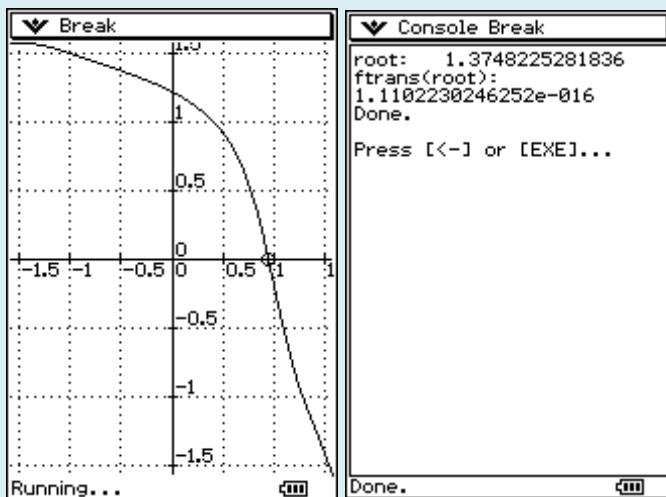
```
require("LNAPlot/PlotFunc","LNAPlot/PlotData","LNA/BrentGh")  
  
local function f(x) return math.exp(1-x)-x/2 end  
  
local function ftrans(x) return math.atan(f(math.tan(x))) end  
  
local root,error,axesdata
```



```
root,error=BrentGhofrani(ftrans,-math.pi/2,math.pi/2)
print("root:",math.tan(root))
print("ftrans(root):",ftrans(root))

axesdata=PlotFunc(ftrans,{-math.pi/2,math.pi/2}, {-math.pi/2,math.pi/2},false,1,"auto",true,1)
PlotData({root},{ftrans(root)},{},axesdata,true,2)
```

Example Program : XBrentGH



Results obtained by the example program **XBrentGH**

Remarks

The interval $\{xl,xr\}$ should contain *exactly* one root. You may need to reduce the desired accuracy only in special cases. The optional argument **maxit** is rarely needed.

FILENAME: **BrentGH**.

DEPENDENCIES: **Epsilon**.

The Driver program code and its requirements code:

```
LNA/BrentGH
```

```
require("LNAutils/Epsilon")

local function NoRoot(fa,fb)
return (fa>0 and fb>0) or (fa<0 and fb<0)
end
```

```

function BrentGhofrani(f,xl,xr,...)
local tol=Epsilon
local maxit=100
local show=false
if arg["n"]>=1 then
  if arg[1]~="auto" then
    tol=arg[1] end
  if arg["n"]>=2 then
    maxit=arg[2]
  if arg["n"]>=3 then
    show=arg[3]
  end
end
end
local root
local eps=Epsilon
local a,b,c,fa,fb,fc,d,e
,tol1,xm,p,q,r,s
if show then print(" i discrepancy") end
a=xl;b=xr;fa=f(a);fb=f(b)
if NoRoot(fa,fb) then
  print("Error: Brent: Interval does not contain a root, or contains an even number of roots.")
  return
end
c=b;fc=fb
for i=1,maxit do
  if NoRoot(fb,fc) then
    c=a;fc=fa;d=b-a;e=d
  end
  if math.abs(fc)<math.abs(fb) then
    a=b;fa=fb;b=c;fb=fc
    c=a;fc=fa
  end
  tol1=2*eps*math.abs(b)+0.5*tol
  xm=0.5*(c-b)
  if show then printf("%2i %+e\n",i,fb) end
  if (math.abs(xm)<=tol1)
  or (fb==0) then
    root=b;return root
  end
  if (math.abs(e)>tol1)
  and (math.abs(fa)>math.abs(fb)) then

```



```

s=fb/fa
if a==c then
  p=2*xm*s;q=1-s
else
  q=fa/fc;r=fb/fc
  p=s*(2*xm*q*(q-r)
    -(b-a)*(r-1))
  q=(q-1)*(r-1)*(s-1)
end
if p>0 then q=-q end
p=math.abs(p)
if 2*p<math.min(3*xm*q-math.abs(tol1*q),math.abs(e*q)) then
  e=d;d=p/q
else
  d=xm;e=d
end
else
  d=xm;e=d
end
a=b;fa=fb
if math.abs(d)>tol1 then
  b=b+d
else
  b=b+math.sign(tol1,xm)
end
fb=f(b)
end
root=b
print("Warning: Brent: Maximum number of iterations reached, but the accuracy criterion is
not satisfied.")
return root
end

export{BrentGhofrani=BrentGhofrani}

```

LNAutils/Epsilon

Epsilon=1.12E-16
 export{Epsilon=Epsilon}



The Example program code and it,s requirements code:

LNAutils/XBrentGh

```
require("LNAPlot/PlotFunc","LNAPlot/PlotData","LNA/BrentGh")

local function f(x) return math.exp(1-x)-x/2 end

local function ftrans(x) return math.atan(f(math.tan(x))) end

local root,error,axesdata

root,error=BrentGhofrani(ftrans,-math.pi/2,math.pi/2)

print("root:",math.tan(root))

print("ftrans(root):",ftrans(root))

axesdata=PlotFunc(ftrans,{math.pi/2},{},"auto",true,1)

PlotData({root},{ftrans(root)},{},axesdata,true,2)
```

LNAutils/PlotFunc

```
require ("table","draw","LNAPlot/PlotUtil")

function PlotFunc(f,xv,yv,...)

--Optional arguments:
```

```
local wait=true  
  
local lwidth={1}  
  
local tics={"auto","auto"}  
  
local grid=true  
  
local lblpos={0,0}  
  
local lblsize={9,9}  
  
local c="auto"  
  
local discont={}  
  
--  
  
local scale  
  
local lws,funcs,x,xp,X,Xp,Y,Yp,disconts  
  
local id={}  
  
if arg["n"]>=1 then  
  
    wait=arg[1]  
  
    if arg["n"]>=2 then  
  
        lwidth=arg[2]  
  
        if type(lwidth)=="number" then  
  
            lwidth={lwidth}  
  
        end  
  
        if arg["n"]>=3 then  
  
            tics=arg[3]  
  
            if type(tics)~="table" then  
  
                tics={tics,tics}
```

```
end

if arg["n"]>=4 then

grid=arg[4]

if arg["n"]>=5 then

lblpos=arg[5]

if type(lblpos)=="number" then

lblpos={lblpos,lblpos}

end

if arg["n"]>=6 then

lblsize=arg[6]

if type(lblsize)=="number" then

lblsize={lblsize,lblsize}

end

if arg["n"]>=7 then

c=arg[7]

if arg["n"]>=8 then

discont=arg[8]

end

end

end

end

end
```

```

end
end

lws=#lwidth

if type(f)=="function" then

f={f}

end

funcs=#f

for i=lws+1,funcs do

lwidth[i]=lwidth[lws]

end

discont,disconts=Discontinuities(funcs,discont)

for i=1,funcs do

id[i]=1

table.sort(discont[i])

end

draw.onbuffer()

if xv[1]~=nil then

scale=SetScale(xv,yv)

PlotAxes(xv,yv,scale,tics,grid,lblpos,lblsize,c)

else

xv=yv[1]

scale=yv[3]

yv=yv[2]

```

```

end

Xp=0;xp=xv[1]

Y={};Yp={}

for i=1,funcs do

Yp[i]=(yv[2]-f[i](xv[1]))*scale[2]

end

for X=1,X_pixels do

x=xv[1]+X/scale[1]

for i=1,funcs do

Y[i]=(yv[2]-f[i](x))*scale[2]

if id[i]>disconts[i] or x<discont[i][id[i]] or xp>discont[i][id[i]] then

draw.line(Xp,Yp[i],X,Y[i],1,lwidth[i])

elseif xp<=discont[i][id[i]] and x>=discont[i][id[i]] then

id[i]=id[i]+1

end

Yp[i]=Y[i]

end

xp=x;Xp=X

end

showgraph()

draw.update()

if wait then waitkey() end

showconsole()

```



```
return {xv,yv,scale}  
end  
  
export{PlotFunc=PlotFunc}
```

LNAutils/PlotUtil

```
require ("string","table","draw","LNAutils/OrderMag")  
  
X_max=158  
Y_max=213  
X_center=79  
Y_center=106  
X_pixels=159  
Y_pixels=214  
  
local function SetScale(xv,yv)  
local xscale,yscale  
xscale=X_pixels/(xv[2]-xv[1])  
yscale=Y_pixels/(yv[2]-yv[1])  
return {xscale,yscale}  
end
```

```

local function AutoXtics(range)
--tics=math.max(math.floor(range/7),1)

local tics=OrderMag(range)

if range<4*tics then

  tics=tics/2

end

return tics

end


local function AutoYtics(range)

--tics=math.max(math.floor((yv[2]-yv[1])/9),1)

local tics=OrderMag(range)

if range<4*tics then

  tics=tics/2

end

return tics

end


function Discontinuities(funcs,discont)

local D=table.copy(discont)

local id={}

local disconts={}

for i=1,funcs do

```

```

if D[i]==nil then
  D[i]={}
elseif type(D[i])=="number" then
  D[i]={D[i]}
end
disconts[i]=#D[i]
end
return D,disconts
end

function PlotAxes(xv,yv,scale,tics,grid,lblpos,lblsize,c)
local Xc,Yc,X,Y,Xl,Yl,lblxp,lblyp
local lblcut=1E-8
--Auto tics selection:
if tics[1]=="auto" then
  tics[1]=AutoXtics(xv[2]-xv[1])
end
if tics[2]=="auto" then
  tics[2]=AutoYtics(yv[2]-yv[1])
end
--
if c=="auto" then
  c={0,0}
end

```

```

end

--[[ AXES ]]

Xc=(c[1]-xv[1])*scale[1]

Yc=(yv[2]-c[2])*scale[2]

draw.line(0,Yc,X_max,Yc)

draw.line(Xc,0,Xc,Y_max)

--[[ TICS & GRID ]]

if lblpos[1]==0 then

Yl=Y_pixels-lblsize[1]

else

Yl=Yc+1

end

if lblpos[2]==0 then

Xl=2

else

Xl=Xc+2

end

for xp=c[1]-math.floor((c[1]-xv[1])/tics[1])*tics[1],math.floor((xv[2]-c[1])/tics[1])*tics[1],tics[1]
do

X=(xp-xv[1])*scale[1]

draw.line(X,Yc-2,X,Yc+2)

if grid then

for Y=0,Y_pixels,4 do

```

```

draw.pixel(X,Y)

end

end

if lblsize[1]>0 and X>0 and X<X_pixels then

lblxp=xp

if math.abs(xp)<=lblcut then

lblxp=0

end

draw.text(X+2,Yl,lblxp,1,lblsize[1])

end

end

for yp=c[2]-math.floor((c[2]-yv[1])/tics[2])*tics[2],math.floor((yv[2]-c[2])/tics[2])*tics[2],tics[2]
do

Y=(yv[2]-yp)*scale[2]

draw.line(Xc-2,Y,Xc+2,Y)

if grid then

for X=0,X_pixels,4 do

draw.pixel(X,Y)

end

end

if lblsize[2]>0 and Y>0 and Y<Y_pixels then

lblyp=yp

if math.abs(yp)<=lblcut then

```

```

lblyp=0

end

draw.text(Xl,Y-lblsize[2],lblyp,1,lblsize[2])

end

end

end

function PlotPoint(X,Y,pointtype,pointsiz
--Non-filled circle:

if pointtype==0 then

draw.point(X,Y,1,pointsiz
elseif pointtype==1 then

draw.circle(X,Y,pointsiz,1,1,-1)

draw.point(X,Y)

--Crossed circle:

elseif pointtype==2 then

draw.circle(X,Y,pointsiz,1,1,-1)

draw.line(X-pointsiz
draw.line(X,Y-pointsiz,X,Y+pointsiz
--Filled circle:

elseif pointtype==3 then

draw.circle(X,Y,pointsiz,1,1,1)

--Non-filled rectangle:

```

```

elseif pointtype==4 then

draw.rect(X-pointsize,Y-pointsize,X+pointsize,Y+pointsize,1,1,-1)

draw.point(X,Y)

--Crossed rectangle:

elseif pointtype==5 then

draw.rect(X-pointsize,Y-pointsize,X+pointsize,Y+pointsize,1,1,-1)

draw.line(X-pointsize,Y,X+pointsize,Y)

draw.line(X,Y-pointsize,X,Y+pointsize)

--Filled rectangle:

elseif pointtype==6 then

draw.rect(X-pointsize,Y-pointsize,X+pointsize,Y+pointsize,1,1,1)

end

end

local function DrawLabel()

return

end

export{X_max=X_max,Y_max=Y_max,X_center=X_center,Y_center=Y_center,X_pixels=X_pixels,
Y_pixels=Y_pixels,SetScale=SetScale,Discontinuities=Discontinuities,PlotAxes=PlotAxes,PlotPoint=PlotPoint}

```

LNAutils/OrderMag



```
require("string")\n\nfunction OrderMag(x)\n    local xexp=string.lower(string.format("%e",x))\n    local e=string.find(xexp,"e")\n    local n=string.len(xexp)\n    return 10^string.sub(xexp,e+1,n)\nend\n\nexport{OrderMag=OrderMag}
```

LNAplot/PlotData

```
require ("table","draw","LNAplot/PlotUtil")\n\nfunction PlotData(x,y,xv,yv,...)\n    --Optional arguments:\n    local wait=true\n    local ptype={1}\n    local psize={3}\n    local lwidth={0}\n    local tics={"auto","auto"}\n    local grid=true
```

```
local lblpos={0,0}  
local lblsize={9,9}  
local c="auto"  
  
--  
  
local points=#x  
local scale  
local sets,pws,lws,X,Xp,Y,Yp  
  
if arg["n"]>=1 then  
    wait=arg[1]  
    if arg["n"]>=2 then  
        ptype=arg[2]  
        if type(ptype)=="number" then  
            ptype={ptype}  
        end  
        if arg["n"]>=3 then  
            psize=arg[3]  
            if type(psize)=="number" then  
                psize={psize}  
            end  
            if arg["n"]>=4 then  
                lwidth=arg[4]  
                if type(lwidth)=="number" then  
                    lwidth={lwidth}
```

```
end

if arg["n"]>=5 then

  tics=arg[5]

  if type(tics)~="table" then

    tics={tics,tics}

  end

  if arg["n"]>=6 then

    grid=arg[6]

    if arg["n"]>=7 then

      lblpos=arg[7]

      if type(lblpos)=="number" then

        lblpos={lblpos,lblpos}

      end

      if arg["n"]>=8 then

        lblsize=arg[8]

        if type(lblsize)=="number" then

          lblsize={lblsize,lblsize}

        end

        if arg["n"]>=9 then

          c=arg[9]

        end

      end

    end

  end

end
```

```
end  
end  
end  
end  
end  
end  
if type(y[1])=="number" then y={y} end  
sets=#y  
pws=#ptype  
for i=pws+1,sets do  
    ptype[i]=ptype[pws]  
end  
pws=#psize  
for i=pws+1,sets do  
    psize[i]=psize[pws]  
end  
lws=#lwidth  
for i=lws+1,sets do  
    lwidth[i]=lwidth[lws]  
end  
draw.onbuffer()  
if xv[1]~=nil then  
    scale=SetScale(xv,yv)
```

```

PlotAxes(xv,yv,scale,tics,grid,lblpos,lblsize,c)

else

xv=yv[1]

scale=yv[3]

yv=yv[2]

end

Xp=(x[1]-xv[1])*scale[1]

Y={};Yp={}

for j=1,sets do

Yp[j]=(yv[2]-y[j][1])*scale[2]

PlotPoint(Xp,Yp[j],ptype[j],psize[j])

end

for i=2,points do

X=(x[i]-xv[1])*scale[1]

for j=1,sets do

Y[j]=(yv[2]-y[j][i])*scale[2]

if psize[j]>0 then

PlotPoint(X,Y[j],ptype[j],psize[j])

end

if lwidth[j]>0 then

draw.line(Xp,Yp[j],X,Y[j],1,lwidth[j])

Yp[j]=Y[j]

end

```

```
end  
  
Xp=X  
  
end  
  
showgraph()  
  
draw.update()  
  
if wait then waitkey() end  
  
showconsole()  
  
return {xv,yv,scale}  
  
end  
  
  
export{PlotData=PlotData}
```