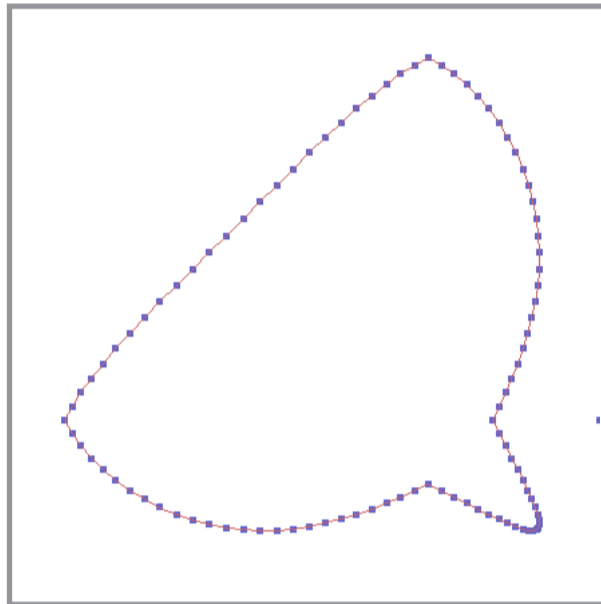


Gernot Hoffmann

Smooth Interpolation for Measured Polygon Contours and for Polylines



Contents

1	Introduction	2
2	Polygon Example	4
3	Polyline Example	6
4	Between Contours	7
5	Polygon Code	8
6	Polyline Code	12
7	Between Contours Code	16

1.1 Introduction

A closed polygon is given by an ordered list of N measured contour points $x(i)$ and $y(i)$, $i=0$ to $N-1$. With increasing index (i) we walk once around the polygon but the data can be garbled.

The functions $x(i)$ and $y(i)$ are converted into two ordinary Fourier series', using a real valued discrete Fourier transformation.

For e.g. $M = 50$ harmonics and the mean value we need $N=2M+1$ samples.

The reconstruction is done by $M=(N \text{ div } 2)$ harmonics or less.

The suppression of higher harmonics smoothes random measurement deviations.

A polyline can be converted into a polygon by a Bézier curve. The polyline is given by e.g. 51 points. The missing 50 points for 101 points altogether are created by the Bézier curve. The last point is the first point.

The tangent at $N/2$ is calculated by the slope of the average line through the last four points (separate for x and y). The tangent at $N-1$ is calculated by the slope of the first four points.

Note: the number of averaged points is called in the program locally also M . This M has nothing to do with M harmonics.

The continuous interpolation along a closed contour is done by using a continuous parameter $s=0$ to 1 for the interval 0 to 2π of the first harmonic (lowest frequency).

The interpolation between two contours is easily done for equally spaced parameters s on both contours by calculating the blended values for $p=0$ (contour 1) to $p=1$ (contour 2):

$$x = (1-p)x_1 + px_2$$

$$y = (1-p)y_1 + py_2$$

Settings for Acrobat

Edit / Preferences / General / Page Display (since version 6)

Custom Resolution 72dpi / **View by zoom 100% or 200%**

Edit / Preferences / General / Color Management (full version only)

sRGB

Euroscale Coated or ISO Coated or SWOP

Gray Gamma 2.2

1.2 Introduction

Formulas

Time domain (t)

Sample interval $dt = T$ for $t=0$ to $(N-1)T$

Lowest frequency $f_1 = 1/(NT)$, $\omega_1 = 2\pi f_1$, $\omega_k = k\omega_1$

Even N $M = N/2$

Odd N $M = N \text{ div } 2$

$$a_k = 2/(NT) \sum_{i=0}^{N-1} f(iT) \cos(k\omega_1 iT)$$

$$b_k = 2/(NT) \sum_{i=0}^{N-1} f(iT) \sin(k\omega_1 iT)$$

$$f(iT) = a_0/2 + \sum_{k=1}^M [a_k \cos(k\omega_1 iT) + b_k \sin(k\omega_1 iT)]$$

Interpolation in the spatial domain for x and y

Parameter $i = 0$ to $N-1$

Continuous Parameter $s = 0$ to 1

Lowest frequency $f_1 = 1/N$, $\omega_1 = 2\pi f_1$, $\omega_k = k\omega_1$

$$a_{xk} = (2/N) \sum_{i=0}^{N-1} x_i \cos(ki2\pi/N)$$

$$b_{xk} = (2/N) \sum_{i=0}^{N-1} x_i \sin(ki2\pi/N)$$

$$x(s) = a_{x0}/2 + \sum_{k=1}^M [a_{xk} \cos(k2\pi s) + b_{xk} \sin(k2\pi s)]$$

$$a_{yk} = (2/N) \sum_{i=0}^{N-1} y_i \cos(ki2\pi/N)$$

$$b_{yk} = (2/N) \sum_{i=0}^{N-1} y_i \sin(ki2\pi/N)$$

$$y(s) = a_{y0}/2 + \sum_{k=1}^M [a_{yk} \cos(k2\pi s) + b_{yk} \sin(k2\pi s)]$$

2.1 Polygon Example

The blue squares represent the measured points. The red polyline shows the reconstruction for different M . Bottom graphic for garbled data.

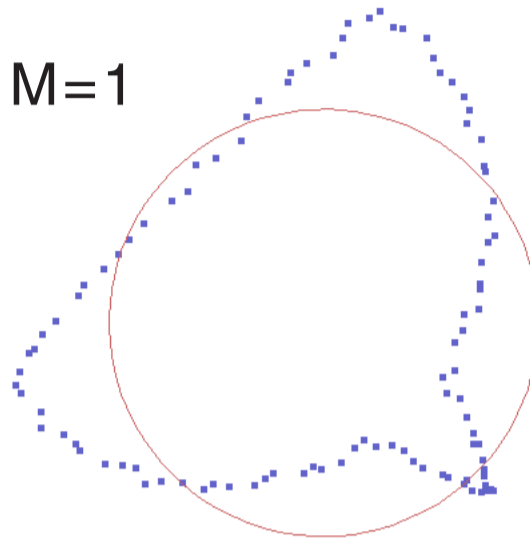
$N = 101$ points

Best view 72dpi / zoom 200%

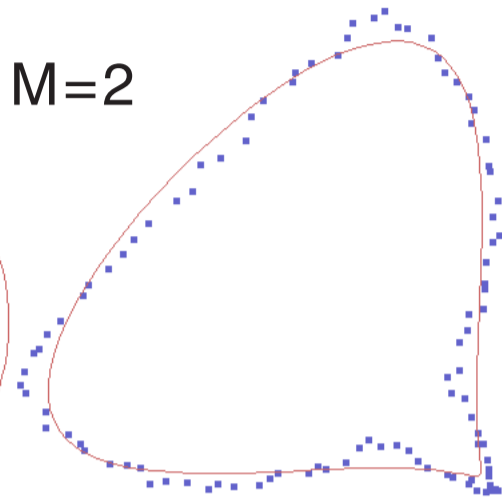
$M=50$



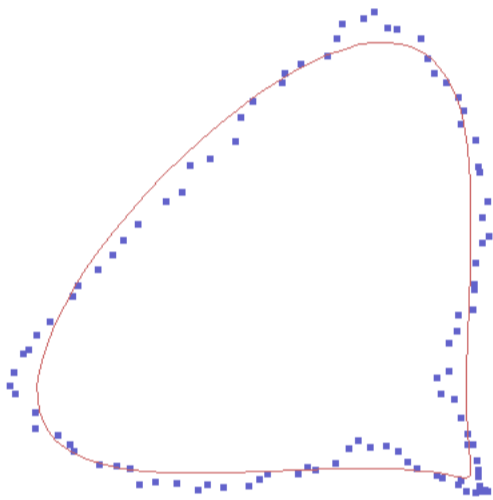
$M=1$



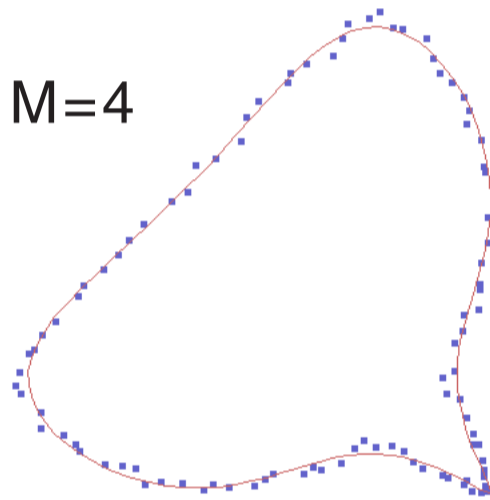
$M=2$



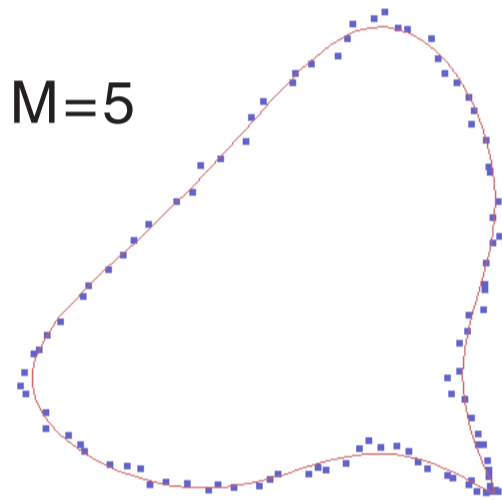
$M=3$



$M=4$



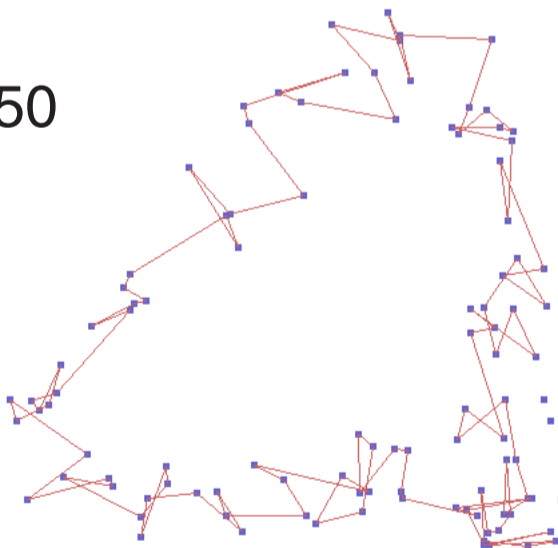
$M=5$



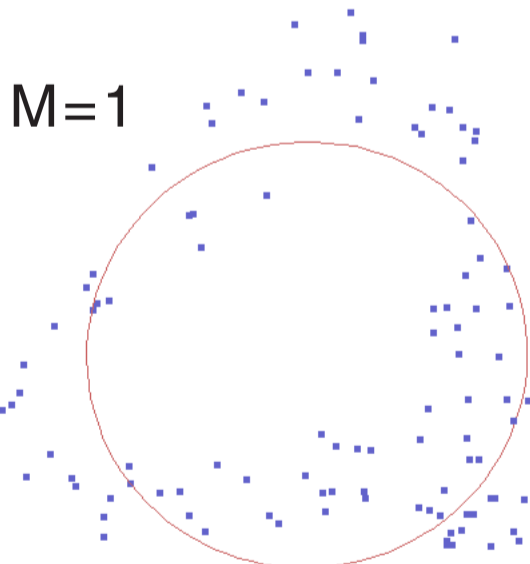
$N = 101$ points

Best view 72dpi / zoom 200%

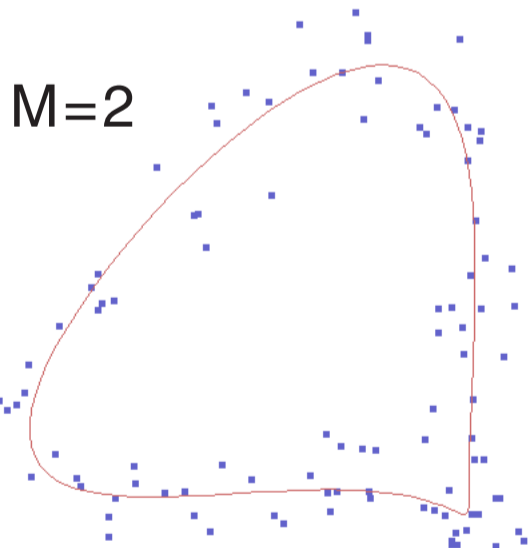
$M=50$



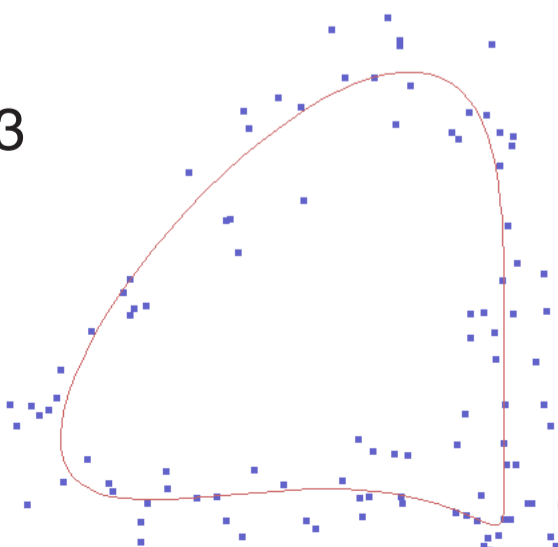
$M=1$



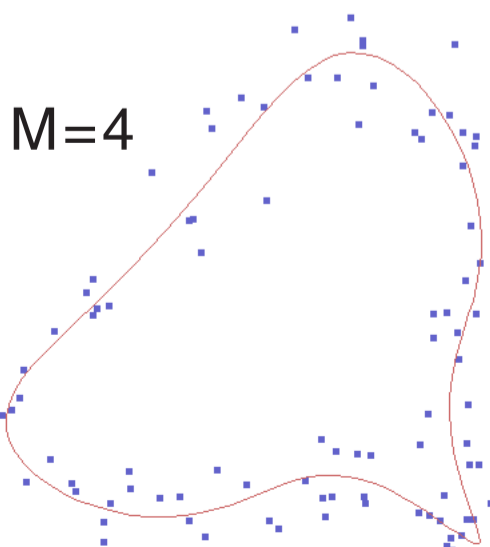
$M=2$



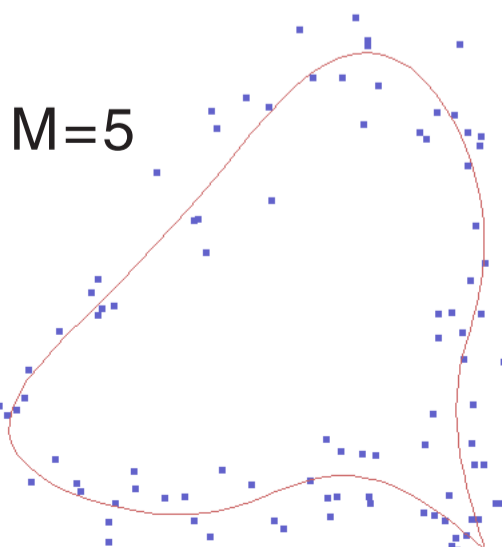
$M=3$



$M=4$



$M=5$

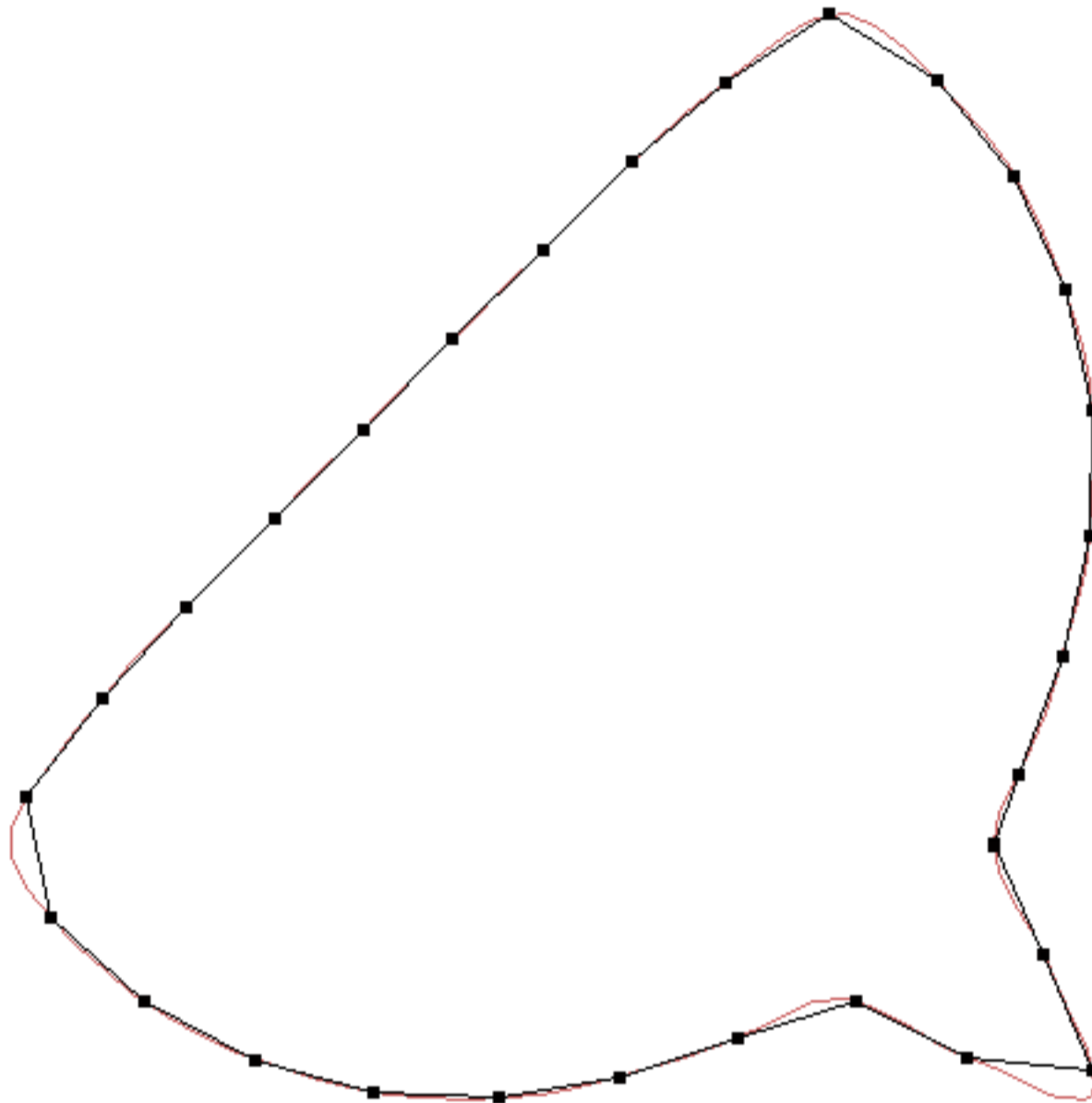


2.2 Polygon Example

Here we see the interpolation by segments of equal lengths. The last segment fills the gap - it is shorter.

A rather crude search by incrementing the parameter s until the segment is long enough.

Best view 72dpi / zoom 100%



3. Polyline Example

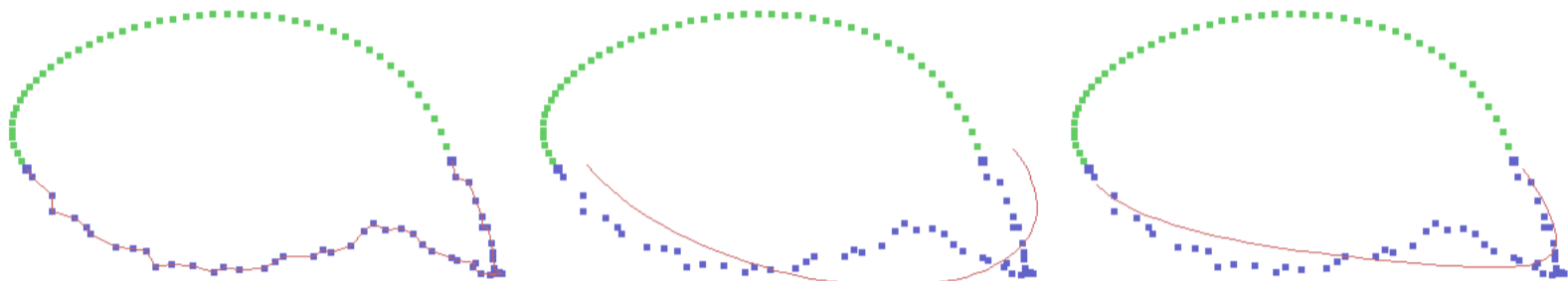
The blue squares represent the measured points. The green dots belong to the Bézier path. The red polyline shows the reconstruction by different values M .

$N = 101$ points (51 points for the polyline) [Best view 72dpi / zoom 200%](#)

$M=50$

$M=1$

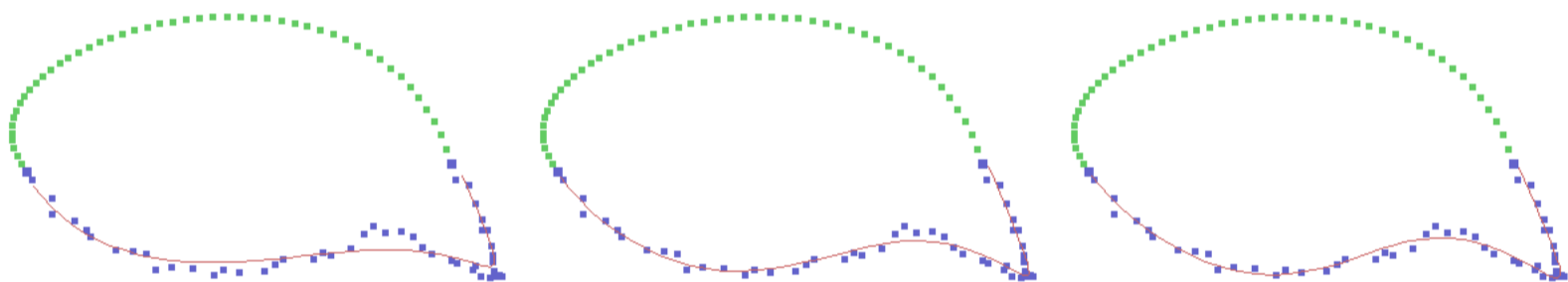
$M=2$



$M=3$

$M=4$

$M=5$



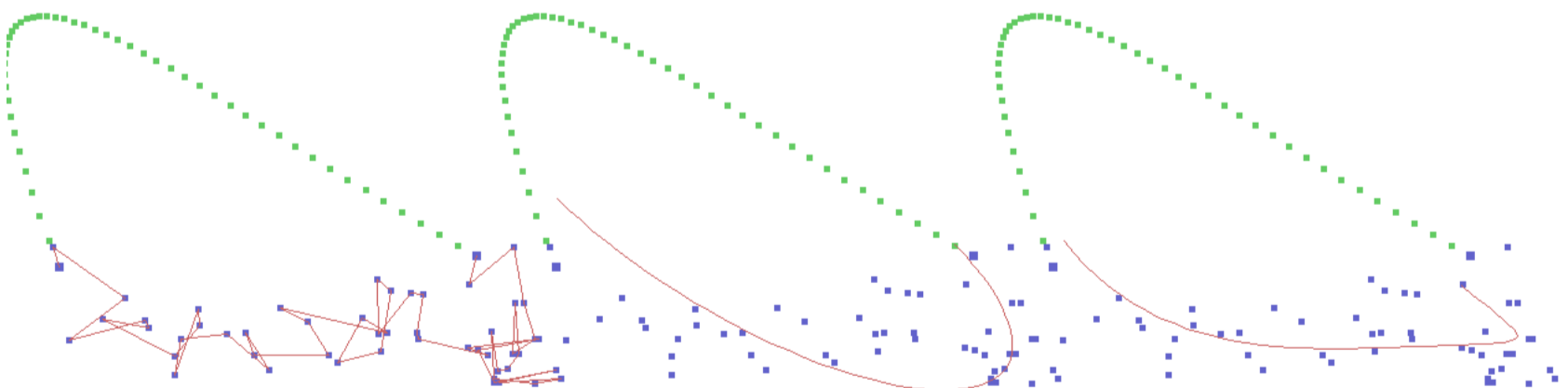
$N = 101$ points

[Best view 72dpi / zoom 200%](#)

$M=50$

$M=1$

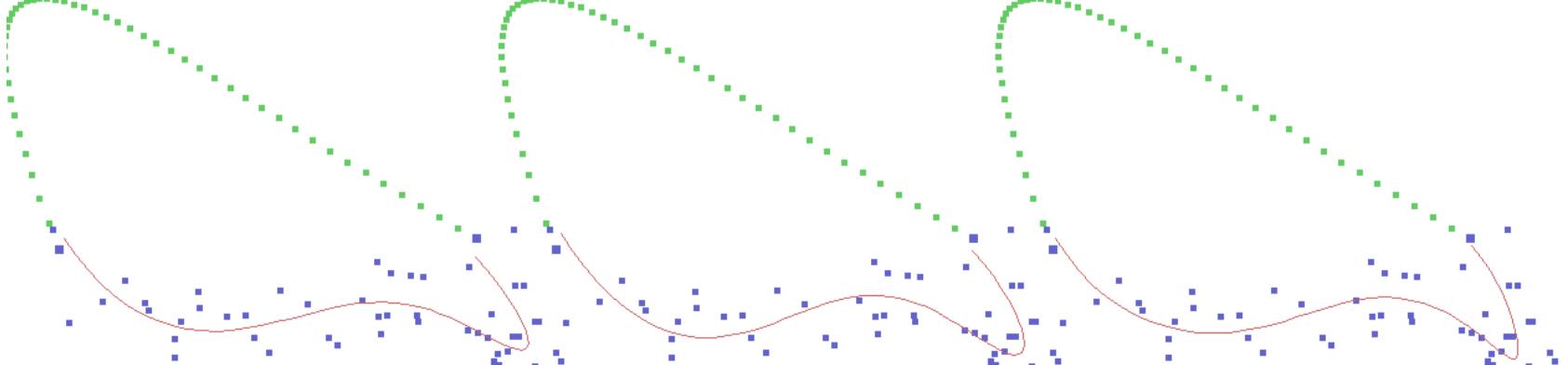
$M=2$



$M=3$

$M=4$

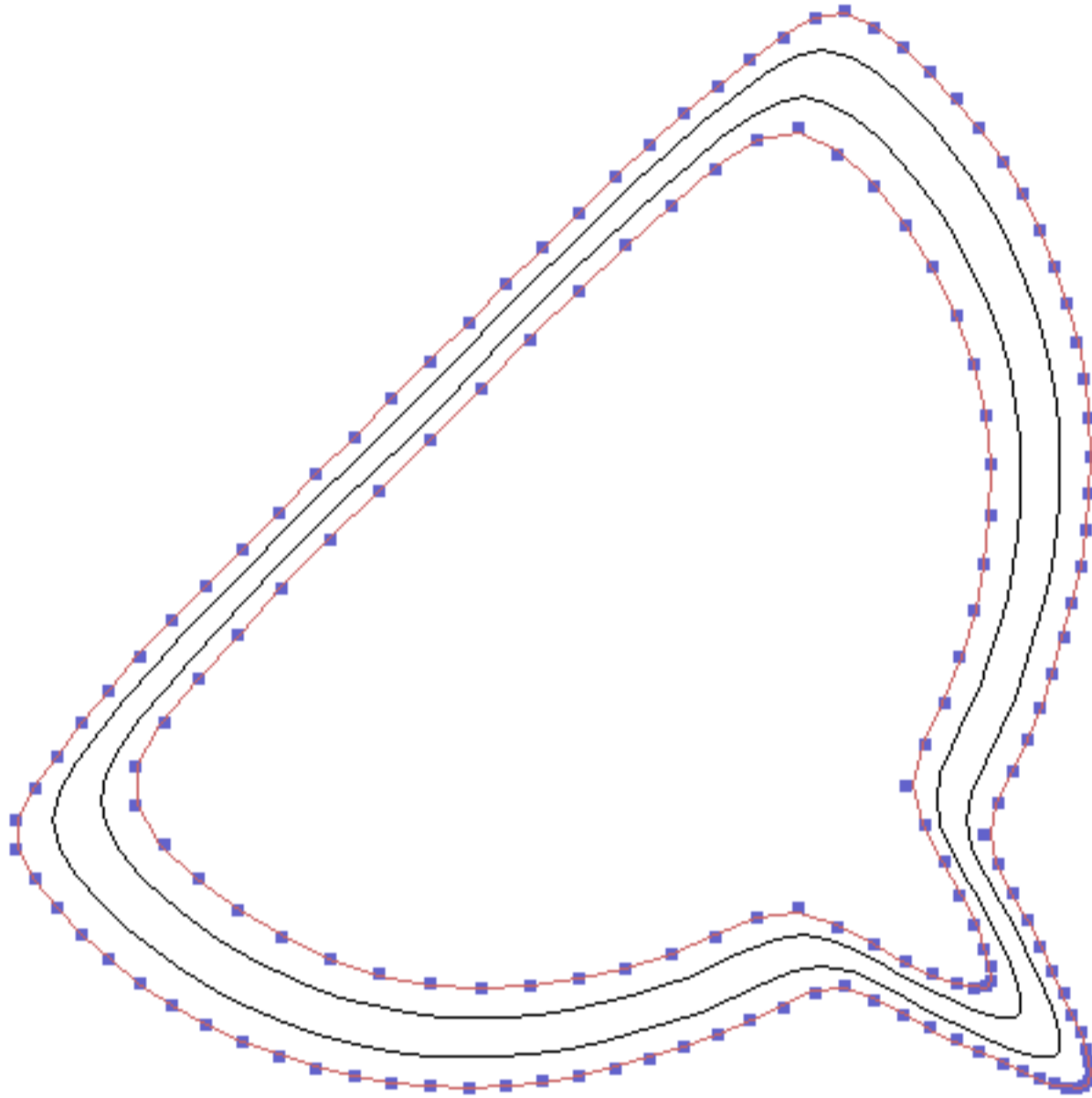
$M=5$



4. Between Contours

The outer curve is made by $N_1 = 101$ points and the inner by $N_2 = 61$. The interpolation by Fourier Series is fairly simple. The interpolated curves are shown for $p = 0.3$ and $p = 0.7$, where $p = 0$ is the outer curve and $p = 1$ the inner.

Best view 72dpi / zoom 100%



5.1 Polygon Code

This Borland Pascal code cannot be used directly, because it is based on a proprietary graphics systems. The code can be used to interpret programming structures.

```
Program ZFPoly02;
{ Project:      Interpolate measured data by Fourier Series
  Author:      G.Hoffmann
  Date:       24.06.2003

{$A+,B-,D-,E-,F-,G+,I+,L+,N+,O-,P-,Q-,R-,S-,T-,V-,X-,Y-}
{$C Moveable PreLoad Permanent }

Uses      Crt,Dos,Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
          Zefir36,Zefir37,Zefir38,Zefir39;
Type      TNarr= Array[0..500] Of Single;
Var       Fx,Fy,Co,Si,Ax,Bx,Ay,By   : TNarr;
Var       N,sc,xp,yp,dx,dy,fi      : Integer;
          Iml                       : String;

Function Ra(dr: Single): Single;
Begin
Ra:=dr*(2*random-1); { -dr..+dr }
End;

Procedure GenDt(N: Integer; Var Fx,Fy: TNarr);
{ Generate data }
Var i: Integer; p2,ss,cc: Single;
Begin
p2:=2*pi/N;
For i:=0 to N-1 Do
  Begin
    SicCoc(i*p2,ss,cc); { Fast Sine+Cosine }
    Fx[i]:=cc+0.5*Abs(ss)-0.2*Sqr(Sqr(cc))+Ra(0.035); { 0.235 }
    Fy[i]:=ss+0.5*Abs(cc)-0.2*Sqr(Sqr(ss))+Ra(0.035); { 0.235 }
  End;
End;

Procedure TDFT(N:Integer; Var Co,Si: TNarr);
{ Sine+Cosine Table }
Var i: Integer; p2: Single;
Begin
p2:=2*pi/N;
For i:=0 to N-1 Do SicCoc(p2*i,Si[i],Co[i]);
End;

Procedure FDFT (N: Integer; Co,Si: TNarr; Var Ax,Bx,FX: TNarr);
{ Forward Fourier Series }
Var at,bt,c2 : Single;
    i,k,ik    : LongInt;
Begin
c2:=2/N;
For k:=0 to N Div 2 Do
Begin
at:=0; bt:=0;
For i:=0 to N-1 Do
Begin
ik:=(i*k) Mod N; { modulo=remainder }
at:=at+Fx[i]*Co[ik];
bt:=bt+Fx[i]*Si[ik];
End; {i}
Ax[k]:=c2*at; Bx[k]:=c2*bt;
End; {k}
End;
```


5.2 Polygon Code

```
Procedure IDFT (N: Integer; Co, Si: TNarr; Var Ax, Bx, Fx: TNarr);
{ Inverse Fourier Series }
Var ft      : Single;
    i, k, ik : LongInt;
Begin
For i:=0 to N-1 Do
Begin
ft:=0.5*Ax[0];
For k:=1 to N div 2 Do
Begin
ik:=(i*k) Mod N; { modulo=remainder }
ft:=ft+Ax[k]*Co[ik]+Bx[k]*Si[ik];
End; {k}
Fx[i]:=ft;
End; {i}
End;

Procedure Mark(x, y, pal, col: Integer);
{ Draw mark }
Var i, j: Integer;
Begin
For i:=-2 to +2 Do
For j:=-2 to +2 Do SpcSixel(x+i, y+j, pal, col);
End;

Procedure DrawM(N, pal, col: Integer);
{ Draw measured points }
Var i: Integer;
Begin
For i:=0 to N-1 Do
Mark(xp+Round(sc*Fx[i]), yp+Round(sc*Fy[i]), pal, col);
End;

Procedure DrawI(N, pal, col: Integer);
{ Draw result of IDFT }
Var i, xo, yo, xn, yn: Integer;
Begin
Fx[N]:=Fx[0]; Fy[N]:=Fy[0]; { close contour }
xo:=xp+Round(sc*Fx[0]); yo:=yp+Round(sc*Fy[0]);
For i:=1 to N Do
Begin
xn:=xp+Round(sc*Fx[i]); yn:=yp+Round(sc*Fy[i]);
MakeSline(xo, yo, xn, yn, pal, col);
xo:=xn; yo:=yn;
End;
End;

Procedure Filt(N: Integer; Var Ax, Bx: TNarr; fi: Integer);
{ No harmonics for k>fi }
Var k: Integer;
Begin
For k:=fi+1 to N Div 2 Do
Begin
Ax[k]:=0; Bx[k]:=0;
End;
End;
```

5.3 Polygon Code

```
Procedure OneShot(xa,ya,fi: Integer);
Begin
Randseed:=0;
xp:=xa+250;
yp:=ya+300;
GenDt(N,Fx,Fy);
DrawM(N,120,150);
TDFT(N,Co,Si);
FDFT(N,Co,Si,Ax,Bx,Fx);
FDFT(N,Co,Si,Ay,By,Fy);
Filt(N,Ax,Bx,fi);
Filt(N,Ay,By,fi);
IDFT(N,Co,Si,Ax,Bx,Fx);
IDFT(N,Co,Si,Ay,By,Fy);
DrawI(N,0,150);
End;

BEGIN
Configure
(ConfigMax,ZebraMax,Flag,Conf,ZebrArray,
 ZebrPath,SmdrPath,
 BuffDrNm,SnapDrNm,ZebrNm,
 LastDriv,SourDriv,DestDriv,
 SourceNr,GcardIdy,VesaMode,VesaCode,
 SoundsOn,NoClkInt,AutoCatG,ShowIcon);
VesaStart(VesaMode);
Im1:='I:\fft__\fft__942.bmp';
ColToScr(grsc,255);           { Background           }
sc:=160;
dx:=400;
dy:=440;
N:=101; { odd ! }
Oneshot( 0, 0,N div 2);
Oneshot( dx, 0,1);
Oneshot(2*dx, 0,2);
Oneshot( 0,dy,3);
Oneshot( dx,dy,4);
Oneshot(2*dx,dy,5);
SaveImag(im1);
Stop;
VesaEnde;
ClrScr;
END.
```

5.4 Polygon Code

Procedure for the interpolation with equal segment lengths

```
Procedure SubDiv(N1,M1: Integer; Ax1,Bx1: TNarr; seg: Single);
{ Interpolation along path with segment length seg }
Var s,ds,x1,y1,x2,y2,A2,B2 : Single;
    i,j,q,xo,yo,xn,yn : Integer;
Procedure IP(s: Single; Var x,y: Single);
{ s=0..1 along path; result x,y }
Var k: Integer; p2,fx1,fy1,si,co: Single;
Begin
    p2:=2*pi*s;
    fx1:=0.5*Ax1[0];
    fy1:=0.5*Ay1[0];
    For k:=1 to M1 Do
    Begin
        SicCoc(p2*k,si,co);
        fx1:=fx1+Ax1[k]*co+Bx1[k]*si;
        fy1:=fy1+Ay1[k]*co+By1[k]*si;
    End; {k}
    x:=fx1;
    y:=fy1;
End;
Begin
B2:=Sqr(seg); { Squared segment length }
ds:=0.001; { smallest step along path }
s:=0;
IP(s,x1,y1);
xo:=Round(sc*x1+xp); yo:=Round(sc*y1+yp);
Mark(xo,yo,2,0,0);
Repeat
    q:=0;
    Repeat
        s:=s+ds;
        IP(s,x2,y2);
        A2:=Sqr(x2-x1)+Sqr(y2-y1);
        Inc(q);
    Until (A2>=B2) Or (q>100) Or (s>=1);
    xn:=Round(sc*x2+xp); yn:=Round(sc*y2+yp);
    MakeSLine(xo,yo,xn,yn,0,0);
    Mark(xn,yn,2,0,0);
    x1:=x2; y1:=y2;
    xo:=xn; yo:=yn;
Until s>=1;
End;
```

6.1 Polyline Code

```
Program ZFPoly01;
{ Project:      Interpolate measured data of a polyline
                  by Bezier and Fourier Series
  Author:      G.Hoffmann
  Date:       July 06, 2003

{$A+,B-,D-,E-,F-,G+,I+,L+,N+,O-,P-,Q-,R-,S-,T-,V-,X-,Y-}
{$C Moveable PreLoad Permanent }

Uses Crt,Dos,Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
     Zefir36,Zefir37,Zefir38,Zefir39;
Type TNarr = Array[0..500] Of Single;
Var  Fx,Fy,Co,Si,Ax,Bx,Ay,By  : TNarr;
Var  N,sc,xp,yp,dx,dy,fi     : Integer;
     Im1                      : String;

Function Ra(dr: Single): Single;
Begin
Ra:=dr*(2*random-1); { -dr..+dr }
End;

Procedure GLine(M,K: Integer; Fx: TNarr; Var a0,a1: Single);
{ Line y=a0+a1*x by M points, starting at K}
Var i: Integer; sx,sy,sx2,sxy,de: Single;
Begin
sx:=0; sx2:=0; sy:=0; sxy:=0;
For i:=0 To M-1 Do
Begin
  sx :=sx +i;
  sx2:=sx2+i*i;
  sy :=sy + Fx[K+i];
  sxy:=sxy+i*Fx[K+i];
End;
de:= M*sx2 -sx*sx;
a0:=(sy*sx2-sx*sxy)/de;
a1:=(M*sxy -sx*sy )/de;
End;

Procedure GenDt(N: Integer; Var Fx,Fy: TNarr);
{ Generate data }
Var  i,n2,M,K: Integer;
     p2,ss,cc,x0,x1,x2,x3,y0,y1,y2,y3,ax,bx,cx,ay,by,cy,t,dt: Single;
     a0,a1,b0,b1: Single;
Begin
n2:=N Div 2;
p2:=2*pi/N;
For i:=0 to n2 Do
  Begin
    SicCoc(i*p2,ss,cc); { Fast Sine+Cosine }
    Fx[i]:=cc+0.5*Abs(ss)-0.2*Sqr(Sqr(cc))+Ra(0.235); { 0.035 }
    Fy[i]:=ss+0.5*Abs(cc)-0.2*Sqr(Sqr(ss))+Ra(0.235); { 0.035 }
  End;
{ Bezier path, using averaged slope for M>=2 points }
M:=4;
K:=n2-M+1;
GLine(M,K,Fx,a0,a1);
GLine(M,K,Fy,b0,b1);
x0:=Fx[n2];
y0:=Fy[n2];
x1:=Fx[n2]+a1*N/6;
y1:=Fy[n2]+b1*N/6;
K:=0;
GLine(M,K,Fx,a0,a1);
GLine(M,K,Fy,b0,b1);
x2:=Fx[ 0]-a1*N/6;
y2:=Fy[ 0]-b1*N/6;
x3:=Fx[ 0];
y3:=Fy[ 0];
```

6.2 Polyline Code

```
cx:=3*(x1-x0); bx:=3*(x2-x1)-cx; ax:=x3-x0-cx-bx;
cy:=3*(y1-y0); by:=3*(y2-y1)-cy; ay:=y3-y0-cy-by;
t:=0; dt:=1/n2;
For i:=n2 to N-1 Do
  Begin
    Fx[i]:=((ax*t+bx)*t+cx)*t+x0;
    Fy[i]:=((ay*t+by)*t+cy)*t+y0;
    t:=t+dt;
  End;
End;

Procedure TDFT (N: Integer; Var Co,Si: TNarr);
{ Sine+Cosine Table }
Var i: Integer; p2: Single;
Begin
  p2:=2*pi/N;
  For i:=0 to N-1 Do SicCoc(p2*i,Si[i],Co[i]);
End;

Procedure FDFT (N: Integer; Co,Si: TNarr; Var Ax,Bx,Fx: TNarr);
{ Forward Fourier Series }
Var at,bt,c2: Single; i,k,ik : LongInt;
Begin
  c2:=2/N;
  For k:=0 to N div 2 Do
  Begin
    at:=0; bt:=0;
    For i:=0 to N-1 Do
    Begin
      ik:=(i*k) Mod N; { modulo=remainder }
      at:=at+Fx[i]*Co[ik];
      bt:=bt+Fx[i]*Si[ik];
    End; {i}
    Ax[k]:=c2*at; Bx[k]:=c2*bt;
  End; {k}
End;

Procedure IDFT (N: Integer; Co,Si: TNarr; Var Ax,Bx,Fx: TNarr);
{ Inverse Fourier Series }
Var ft : Single;
  i,k,ik: LongInt;
Begin
  For i:=0 to N-1 Do
  Begin
    ft:=0.5*Ax[0];
    For k:=1 to N div 2 Do
    Begin
      ik:=(i*k) Mod N; { modulo=remainder }
      ft:=ft+Ax[k]*Co[ik]+Bx[k]*Si[ik];
    End; {k}
    Fx[i]:=ft;
  End; {i}
End;
```

6.3 Polyline Code

```
Procedure Mark(x,y,k,pal,col: Integer);
{ Draw mark }
Var i,j: Integer;
Begin
For i:=-k to +k Do
For j:=-k to +k Do SpcSixel(x+i,y+j,pal,col);
End;

Procedure DrawM(N,p1,c1,p2,c2: Integer);
{ Draw measured points }
Var i,n2: Integer;
Begin
n2:=N div 2;
Mark(xp+Round(sc*Fx[ 0]),yp+Round(sc*Fy[ 0]),3,p1,c1);
For i:=0 to n2-1 Do
Mark(xp+Round(sc*Fx[ i]),yp+Round(sc*Fy[ i]),2,p1,c1);
Mark(xp+Round(sc*Fx[n2]),yp+Round(sc*Fy[n2]),3,p1,c1);
For i:=n2+1 to N-2 Do
Mark(xp+Round(sc*Fx[ i]),yp+Round(sc*Fy[ i]),2,p2,c2);
End;

Procedure DrawI(N,pal,col: Integer);
{ Draw result of IDFT }
Var i,xo,yo,xn,yn,n2: Integer;
Begin
n2:=N div 2;
xo:=xp+Round(sc*Fx[0]); yo:=yp+Round(sc*Fy[0]);
For i:=1 to n2 Do
Begin
xn:=xp+Round(sc*Fx[i]); yn:=yp+Round(sc*Fy[i]);
MakeSline(xo,yo,xn,yn,pal,col);
xo:=xn; yo:=yn;
End;
End;

Procedure Filt(N: Integer; Var Ax,Bx: TNarr; fi: Integer);
{ No harmonics for k>fi }
Var k: Integer;
Begin
For k:=fi+1 to N Div 2 Do
Begin
Ax[k]:=0; Bx[k]:=0;
End;
End;

Procedure OneShot(xa,ya,fi: Integer);
Begin
xp:=xa+250;
yp:=ya+300;
Randseed:=0;
GenDt(N,Fx,Fy);
DrawM(N,120,150,60,150);
TDFT (N,Co,Si);
FDFT (N,Co,Si,Ax,Bx,Fx);
FDFT (N,Co,Si,Ay,By,Fy);
Filt (N,Ax,Bx,fi);
Filt (N,Ay,By,fi);
IDFT (N,Co,Si,Ax,Bx,Fx);
IDFT (N,Co,Si,Ay,By,Fy);
DrawI(N,0,150);
End;
```

6.4 Polyline Code

```
BEGIN
Configure
(ConfigMax, ZebraMax, Flag, Conf, ZebrArray,
 ZebrPath, SmdrPath,
 BuffDrNm, SnapDrNm, ZebrNm,
 LastDriv, SourDriv, DestDriv,
 SourceNr, GcardIdy, VesaMode, VesaCode,
 SoundsOn, NoClkInt, AutoCatG, ShowIcon);
VesaStart(VesaMode);
Im1:='I:\fft__\fft__944.bmp';
ColToScr(grsc,255);
sc:=160;
dx:=400;
dy:=440;
N:=101; { odd }
Oneshot( 0, 0,N div 2);
Oneshot( dx, 0,1);
Oneshot(2*dx, 0,2);
Oneshot( 0,dy,3);
Oneshot( dx,dy,4);
Oneshot(2*dx,dy,5);
SaveImag(im1);
Stop;
VesaEnde;
ClrScr;
END.
```

7.1 Between Contours Code

```
Program ZFPoly03;
{ Project:      Interpolate between two closed contours
                  by Fourier Series
  Author:      G.Hoffmann
  Date:        July 10,2003

{$A+,B-,D-,E-,F-,G+,I+,L+,N+,O-,P-,Q-,R-,S-,T-,V-,X-,Y-}
{$C Moveable PreLoad Permanent }

Uses      Crt,Dos,Zefir30,Zefir31,Zefir32,Zefir33,Zefir34,Zefir35,
          Zefir36,Zefir37,Zefir38,Zefir39;
Type      TNarr = Array[0..500] Of Single;
Var        Fx1,Fy1,Co1,Si1,Ax1,Bx1,Ay1,By1: TNarr;
          Fx2,Fy2,Co2,Si2,Ax2,Bx2,Ay2,By2: TNarr;
Var        N1,N2,M1,M2,sc,yp,fp,f2      : Integer;
          Im1                            : String;
          p                               : Single;

Procedure GenDt(N,P: Integer; Var Fx,Fy: TNarr);
{ Generate data }
Var i      : Integer;
    p2,ss,cc : Single;
    x0,x1,x2,x3,y0,y1,y2,y3,ax,bx,cx,ay,by,cy,t,dt: Single;
Begin
p2:=2*pi/N;
Case P Of
1: For i:=0 to N-1 Do
    Begin
        SicCoc(i*p2,ss,cc); { Fast Sine+Cosine }
        Fx1[i]:=cc+0.5*Abs(ss)-0.2*Sqr(Sqr(cc));
        Fy1[i]:=ss+0.5*Abs(cc)-0.2*Sqr(Sqr(ss));
    End;
2: For i:=0 to N-1 Do
    Begin
        SicCoc(i*p2,ss,cc); { Fast Sine+Cosine }
        Fx2[i]:=0.8*(cc+0.5*Abs(ss)-0.2*Sqr(Sqr(cc)));
        Fy2[i]:=0.8*(ss+0.5*Abs(cc)-0.2*Sqr(Sqr(ss)));
    End;
End; { Case}
End;

Procedure TDFT (N: Integer; Var Co,Si: TNarr);
{ Sine+Cosine Table }
Var i: Integer; p2: Single;
Begin
p2:=2*pi/N;
For i:=0 to N-1 Do SicCoc(p2*i,Si[i],Co[i]);
End;

Procedure FDFT (N: Integer; Co,Si: TNarr; Var Ax,Bx,Fx: TNarr);
{ Forward Fourier Series }
Var at,bt,c2 : Single;
    i,k,ik    : LongInt;
Begin
c2:=2/N;
For k:=0 to N div 2 Do
Begin
at:=0; bt:=0;
For i:=0 to N-1 Do
Begin
ik:=(i*k) Mod N; { modulo=remainder }
at:=at+Fx[i]*Co[ik];
bt:=bt+Fx[i]*Si[ik];
End; {i}
Ax[k]:=c2*at; Bx[k]:=c2*bt;
End; {k}
End;
```


7.2 Between Contours Code

```
Procedure IDFT (N: Integer; Co, Si: TNarr; Var Ax, Bx, Fx: TNarr);
{ Inverse Fourier Series }
Var ft      : Single;
    i, k, ik : LongInt;
Begin
For i:=0 to N-1 Do
Begin
ft:=0.5*Ax[0];
For k:=1 to N div 2 Do
Begin
ik:=(i*k) Mod N; { modulo=remainder }
ft:=ft+Ax[k]*Co[ik]+Bx[k]*Si[ik];
End; {k}
Fx[i]:=ft;
End; {i}
End;

Procedure Mark(x, y, k, pal, col: Integer);
{ Draw mark }
Var i, j: Integer;
Begin
For i:=-k to +k Do
For j:=-k to +k Do SpcSixel(x+i, y+j, pal, col);
End;

Procedure DrawM(N, xp, yp, p1, c1, p2, c2: Integer; Fx, Fy: TNarr);
{ Draw measured points }
Var i: Integer;
Begin
For i:=0 to N-1 Do
Mark(xp+Round(sc*Fx[ i]), yp+Round(sc*Fy[ i]), 2, p1, c1);
End;

Procedure DrawI(N, pal, col: Integer; Fx, Fy: TNarr);
{ Draw result of IDFT }
Var i, xo, yo, xn, yn: Integer;
Begin
Fx[N]:=Fx[0]; Fy[N]:=Fy[0];
xo:=xp+Round(sc*Fx[0]); yo:=yp+Round(sc*Fy[0]);
For i:=1 to N Do
Begin
xn:=xp+Round(sc*Fx[i]); yn:=yp+Round(sc*Fy[i]);
MakeSline(xo, yo, xn, yn, pal, col);
xo:=xn; yo:=yn;
End;
End;

Procedure Filt(N, M: Integer; Var Ax, Bx: TNarr);
{ No harmonics for k>fi }
Var k: Integer;
Begin
For k:=M+1 to N Div 2 Do
Begin
Ax[k]:=0; Bx[k]:=0;
End;
End;
```

7.3 Between Contours Code

```
Procedure IntPol(N1,N2,M1,M2: Integer; Ax1,Bx1,Ax2,By2: TNarr; p: Single);
{ p=0..1 : Interpolation }
Var i,j,q : Integer;
    s,ds,fx1,fy1,fx2,fy2,fx,fy,a,si,co : Single;
    x,y,p2 : Single;
    xo,yo,xn,yn : Integer;
Procedure IP;
{ s=0..1 along path; result x,y }
Var k: Integer;
Begin
    p2:=2*pi*s;
    fx1:=0.5*Ax1[0];
    fy1:=0.5*Ay1[0];
    For k:=1 to M1 Do
    Begin
        a:=p2*k;
        SicCoc(a,si,co);
        fx1:=fx1+Ax1[k]*co+Bx1[k]*si;
        fy1:=fy1+Ay1[k]*co+By1[k]*si;
    End; {k}
    fx2:=0.5*Ax2[0];
    fy2:=0.5*Ay2[0];
    For k:=1 to M2 Do
    Begin
        a:=p2*k;
        SicCoc(a,si,co);
        fx2:=fx2+Ax2[k]*co+Bx2[k]*si;
        fy2:=fy2+Ay2[k]*co+By2[k]*si;
    End; {k}
    x:=(1-p)*fx1+p*fx2;
    y:=(1-p)*fy1+p*fy2;
End;
Begin
q :=100; { steps along path }
ds:=1/q;
s:=0;
IP;
xo:=Round(sc*x+xp); yo:=Round(sc*y+yp);
s:=s+ds;
For j:=1 to q Do
Begin
    IP;
    xn:=Round(sc*x+xp); yn:=Round(sc*y+yp);
    MakeSLine(xo,yo,xn,yn,0,0);
    xo:=xn; yo:=yn;
    s:=s+ds;
End;
End;

BEGIN
Configure
(ConfigMax,ZebraMax,Flag,Conf,ZebrArray,
 ZebrPath,SmdrPath,
 BuffDrNm,SnapDrNm,ZebrNm,
 LastDriv,SourDriv,DestDriv,
 SourceNr,GcardIdy,VesaMode,VesaCode,
 SoundsOn,NoClkInt,AutoCatG,ShowIcon);
VesaStart(VesaMode);
Iml:='I:\fft_\fft__991.bmp';
ColToScr(grsc,255); { Background }
sc:=200;
xp:=300;
yp:=300;
N1:=101; { odd }
N2:= 61;
M1:=N1 div 5;
M2:=N2 div 5;
```

7.4 Between Contours Code

```
GenDt (N1, 1, Fx1, Fy1);
DrawM(N1, xp, yp, 120, 150, 60, 150, Fx1, Fy1);
TDFT (N1, Co1, Si1);
FDFT (N1, Co1, Si1, Ax1, Bx1, Fx1);
FDFT (N1, Co1, Si1, Ay1, By1, Fy1);
Filt (N1, M1, Ax1, Bx1);
Filt (N1, M1, Ay1, By1);
IDFT (N1, Co1, Si1, Ax1, Bx1, Fx1);
IDFT (N1, Co1, Si1, Ay1, By1, Fy1);
DrawI (N1, 0, 150, Fx1, Fy1);
GenDt (N2, 2, Fx2, Fy2);
DrawM(N2, xp, yp, 120, 150, 60, 150, Fx2, Fy2);
TDFT (N2, Co2, Si2);
FDFT (N2, Co2, Si2, Ax2, Bx2, Fx2);
FDFT (N2, Co2, Si2, Ay2, By2, Fy2);
Filt (N2, M2, Ax2, Bx2);
Filt (N2, M2, Ay2, By2);
IDFT (N2, Co2, Si2, Ax2, Bx2, Fx2);
IDFT (N2, Co2, Si2, Ay2, By2, Fy2);
DrawI (N2, 0, 150, Fx2, Fy2);
p:=0.3;
IntPol (N1, N2, M1, M2, Ax1, Bx1, Ax2, By2, p);
p:=0.7;
IntPol (N1, N2, M1, M2, Ax1, Bx1, Ax2, By2, p);
SaveImag(im1);
Stop;
VesaEnde;
ClrScr;
END.
```