DELFT UNIVERSITY OF TECHNOLOGY
Department of Electrical Engineering
Telecommunications and
Traffic-Control Systems Group.

Title	:	A NOVEL TECHNIQUE FOR HANDWRITTEN CHARACTER RECOGNITION USING GENERALISED FOURIER DESCRIPTORS
Author	:	L.P.W. Niemel.

Type	:	Graduation thesis
Number of pages	:	62 + vii
Date	:	June 21, 1991.

Graduate Professor	:	Prof. dr. J.C. Arnbak
Mentors	:	Prof. dr. R. Prasad and ir. N.B.J. Weyland.
Assignment number	:	A - 389
Period	:	October 1990 - June 1991

In this graduation thesis, a new method is presented for recognition of handwritten characters. Character recognition is done on the basis of Generalised Fourier Descriptors. The characters are first fitted into straight line sections and arcs of circles, because expressions for the Generalised Fourier descriptors of these basic curves are known. The Generalised Fourier Descriptors are calculated for different Western-style characters. Also different handwritings are involved to compare the effect of difference in writing styles.

Indexing terms : Pattern recognition, character recognition.

SUMMARY

Character recognition is an important tool in communication and computer practice. Handwritten characters are curves that can very efficiently be coded by considering the difference of tangents along the curve. This coding technique was developed at Delft University of Technology and is known as Differential Chain Coding (DCC). A character recognition method which combines well with DCC codes is Generalised Fourier Descriptors (GFDs). Mathematical expressions have been found when GFDs are applied on two important curves, namely straight line pieces and arcs of circles.

In this report, these results have been applied on handwritten characters. Of course, man's handwriting generally does not consist of circles and straight lines. In order to apply the results found for straight line pieces and arcs of circles on handwritten characters, the characters were first split up into parts that are curve-fitted to the closest straight line or circle. Finally, calculation of GFDs has been performed.

Some experiments have been done, including several distinct characters as well as several different handwritings, in order to show the use of this method. From these experiments it can be concluded that GFDs show good distinguishing properties for different characters of the Western alphabet. When the GFDs are applied on different handwritings, they appear to be quite similar.

LIST OF SYMBOLS AND ABBREVIATIONS

FDs	: Fourier Descriptors.
GFDs	: Generalised Fourier Descriptors.
DCC	: Differential Chain Coding.
LS	: Least Squares.
TLS	: Total Least Squares.
SVD	: Singular Value Decomposition.
γ	: denotes a handwritten curve.
$\theta(s)$	: absolute tangent of the curve γ at a distance s from starting point.
s	: nominal distance from starting point along the curve γ.
L	: Total nominal length of the curve γ.
t	: Normalised distance parameter, such that $0 \leq t \leq 1$.
$\phi(s)$	: Relative tangent of the curve γ at distance s from starting point.
$\phi^*(t)$	: Normalised relative tangent of the curve γ at normalised distance t from starting point.
ξ	: linear correcting constant, used to normalise $\phi(s)$ to $\phi^*(t)$.
Z(l)	: complex constant representing combined X-jY-coordinates at distance l.
X(l)	: real part of Z(l), representing X-coordinate at distance l.
Y(l)	: imaginary part of Z(l), representing Y-coordinate at distance l.
$\psi(t)$	: Periodic expansion of the normalised function $\phi^*(t)$.
μ_0	: DC-component in the spectrum of $\psi(t)$.
a_k, b_k	: Fourier Coefficients at sample frequency k, where $k > 0$.
A_k	: Spectral amplitude.
α_k	: Spectral phase.
$\kappa(s)$	: curvature of the curve γ at distance s from starting point.
$\kappa^*(t)$	: Normalised curvature at normalised distance t from starting point.
$\Phi(f)$	: Fourier Transform of the function $\phi(s)$.
$\Phi^*(f)$	: Fourier Transform of the function $\phi^*(t)$.
K(f)	: Fourier Transform of function $\kappa(s)$.
$K^*(f)$	: Fourier Transform of function $\kappa^*(t)$.
W	: Spectral bandwidth.
f_s	: sampling frequency in t-domain.
T	: Sampling period in t-domain.

N_t	: Number of samples in t-domain.
N_f	: Number of samples in f-domain.
m_x	: X-coordinate of centre of a circle.
m_y	: Y-coordinate of centre of a circle.
r	: radius of a circle.

CONTENTS

CHAPTER 1. INTRODUCTION

Character recognition is becoming an important tool in communication practice. (An overview of research in this field can be found in [15]). In typed character communication teletext, for example, character recognition can be used to recover distorted characters, which have suffered from transmission errors or datacompression [8]. Fixed-font communication however is a rather inefficient way of transmitting information. Instead of transmitting the whole plane of pixels such as in facsimile, it would be more efficient to transmit only the information of the character itself. In telewriting (Prasad, Vieveen, Bons and Arnbak [5]), a writing tablet is used to transmit curves of arbitrary shape, such as maps, drawings, handwritten messages and so on. At Delft University of Technology, Differential Chain Coding (DCC) was developed for line-drawings in order to achieve a better coding efficiency. It has been established that DCC is a highly efficient code for line-drawings [5], [6] and graphics [3].

Character recognition methods which combine well with DCC-codes are Fourier Descriptors (Zahn and Roskies[1], Freeman[2]) which are only applicable on simple, clockwise oriented closed curves, and Generalised Fourier Descriptors (GFDs) [4], which can be applied on arbitrary curves such as handwritten characters. Weyland and Prasad ([7],[16]) have given a more suitable definition of GFDs. These are all deterministic methods, which make them convenient for mathematical manipulation. Throughout this work, GFDs as defined in [7] and [16] will be used for character recognition. Weyland and Prasad have also derived analytical expressions to obtain GFDs for two basic types of curves, namely straight lines and circles.

Of course, man's handwritten text seldom consists of arcs of circles and straight-line segments. In this work we try to apply these results on real handwritten characters. This involves a decision to make a suitable choice of data (namely, differential angles versus X-Y-coordinates), a correct partitioning of the character into pieces that are approximated by either straight line-pieces or arcs of circles, the approximation itself and finally calculating the GFDs. At this point we can perform calculation on all characters of the alphabet and start character recognition.

Chapter 2 gives an introduction to Generalised Fourier Descriptors. In chapter 3, data-fitting is discussed, using angular data. In chapter 4, curve-fitting techniques are discussed for use on X-Y-coordinates, which will appear to be more profitable. Based on these X-Y-coordinates, curve-fitting will be performed on handwritten characters in chapter 5. In chapter 6 we will calculate the GFDs for the approximated curves for several characters and several handwritings. Also a character recognition frame will be introduced. Finally, in chapter 7 conclusions are drawn from these character recognition results. Appendix A gives a short introduction to calculation of the Singular Value Decomposition, and appendix B gives the complete listings of the Pascal programs written during the graduation work.

CHAPTER 2. THE CONCEPT OF GENERALISED FOURIER DESCRIPTORS

2.1. INTRODUCTION

In this chapter the various existing Fourier Descriptors will be discussed. The key idea in all types of Fourier Descriptors is to apply the Fourier transform on a written curve. The resulting spectrum is then expected to possess certain characteristics for each letter from the alphabet. First, the Fourier Descriptors as introduced by Zahn and Roskies [1] will be discussed. This definition is only valid for simple, clockwise oriented, closed curves. In order to make the concept of Fourier Descriptors also valid for the more common class of non-simple, non-clockwise, open curves (such as handwritten patterns), Generalised Fourier descriptors were defined by Vieveen and Prasad [4]. This definition was later modified by Weyland and Prasad to overcome some restrictions [7]. The latter definition will be used in succeeding chapters.

2.2. FOURIER DESCRIPTORS (FDs).

Let γ denote a curve in the X-Y-plane. Figure 1 illustrates such a curve with the important parameters. The idea is not to describe the curve in terms of X-Y-coordinates, but to use the angle $\theta(s)$ of the curve at a distance s from the starting point. In this way, the curve γ is uniquely defined with the angle parameter θ and the distance parameter s, the so-called natural parameters.

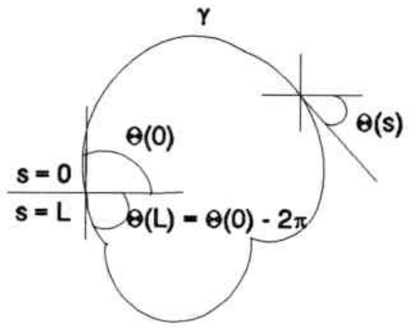


Figure 1 Arbitrary closed, simple curve with important parameters

The total length of the curve is denoted as L. θ(s) denotes the absolute angle of the curve, at a distance s from the starting point. s lies in the range $0 \leq s \leq L$, where L denotes the total length of the curve. It is more efficient to use the angular difference function ϕ(s) instead, which is defined as :

$$\phi(s) \triangleq \theta(s) - \theta(0) \tag{2-1}$$

We observe here that all simple, clockwise oriented closed curves have $\phi(L)=-2\pi$. since the initial angle coincides with the final angle. The distance parameter can be normalised to the interval $[0, 2\pi]$ by introducing $t=2\pi s/L$. Also the phase function is modified in order to let initial angle coincide with the final angle, by defining :

$$\phi^*(t) \triangleq \phi(\frac{Lt}{2\pi}) + t \tag{2-2}$$

Zahn and Roskies [1] have proved that the X-Y-coordinates can be retrieved from the angular function according to the so-called reconstruction theorem:

$$Z(l) = Z(0) + \int_0^l e^{j\theta(\lambda)}\, d\lambda \tag{2-3}$$

which is, since Z = X + jY, equivalent to :

$$\begin{aligned} x(l) &= x(0) + \int_0^l \cos\, \theta(\lambda)\, d\lambda \\ y(l) &= y(0) + \int_0^l \sin\, \theta(\lambda)\, d\lambda \end{aligned} \tag{2-4}$$

We will now try to expand this function into a Fourier series. A Fourier series, however, can be only obtained for periodic functions. Since ϕ^*(t) is not periodic (it is only defined in for $0 \leq t \leq 2\pi$), a periodic version will therefore be made of ϕ^*(t) as follows :

$$\psi(t) \triangleq \sum_{k=-\infty}^{\infty} \phi^*(t-kT) \qquad kT \leq t \leq (k+1)T \tag{2-5}$$

where k= ...-3,-2,-1.0,1,2,3,... and $T \leq 2\pi$. This periodic function can be expanded into a Fourier series as follows :

$$\begin{aligned}\psi(t) &= \mu_0 + \sum_{k=1}^{\infty} (a_k \cos kt + b_k \sin kt) \\ &= \mu_0 + \sum_{k=1}^{\infty} A_k \cos(kt - \alpha_k)\end{aligned} \qquad (2\text{-}6)$$

where (A_k, α_k) are the polar coordinates of (a_k, b_k). A_k represents the kth harmonic amplitude and α_k represents the kth harmonic phase of the spatial frequency spectrum. According to Zahn and Roskies [1] A_k and α_k are the Fourier Descriptors (FDs) of the curve γ. The following relations hold between the FDs and the Fourier coefficients of the curve γ :

$$A_k = \sqrt{a_k^2 + b_k^2} \quad , \quad \tan \alpha_k = b_k / a_k \qquad (2\text{-}7)$$

In order to calculate the FDs, the values of the Fourier coefficients a_k and b_k must be calculated first. The following relations hold for the Fourier coefficients :

$$\mu_0 = \frac{1}{2\pi} \int_0^{2\pi} \phi^*(t)\, dt \qquad (2\text{-}8a)$$

$$a_n = \frac{1}{\pi} \int_0^{2\pi} \phi^*(t) \cos nt\, dt \qquad (2\text{-}8b)$$

$$b_n = \frac{1}{\pi} \int_0^{2\pi} \phi^*(t) \sin nt\, dt \qquad (2\text{-}8c)$$

At this point all the formulae are derived to calculate the FDs A_k and α_k for the curve γ. In practice, however, we do not have the actual function $\phi^*(t)$ at our disposal, but a sampled angular function which comes from the writing tablet, on which characters are written for experiments. Since the writing tablet has a certain resolution, the angular data can be considered as a sampled version of the actual phase function. When the curve γ is, by the sampling process, divided into m sections, we have the m different straight line-pieces l_k and corresponding angle sections $\Delta\phi_i$. Vieveen and Prasad [4] have proven that the following relations hold :

$$\mu_0 = -\pi - \frac{1}{L} \sum_{k=1}^{m} l_k \Delta\phi_k \qquad (2\text{-}9a)$$

$$a_n = -\frac{1}{n\pi}\sum_{k=1}^{m} \Delta\phi_k \sin\frac{2\pi n l_k}{L} \qquad \text{(2-9b)}$$

$$b_n = \frac{1}{n\pi}\sum_{k=1}^{m} \Delta\phi_k \cos\frac{2\pi n l_k}{L} \qquad \text{(2-9c)}$$

So far, the formulae of importance and as defined by Zahn and Roskies [1] are discussed. For applications of these formulae in order to retrieve shape information and similarity functions, we refer to the paper by Zahn and Roskies [1]. In their article, also experiments are performed, which demonstrate the usefulness of the FDs for recognition purposes.

2.3. GENERALISED FOURIER DESCRIPTORS (GFDs).

As mentioned earlier, FDs as discussed in the previous paragraph, are only applicable on simple, closed, clockwise oriented curves. Of course, for character recognition purposes, this is a severe restriction, since most alphabetic characters are often non-simple, open, non-clockwise oriented curves, or combinations of these. Therefore, Generalised Fourier Descriptors (GFDs) were defined by Vieveen and Prasad [4]. A better definition was later given by Weyland and Prasad [7]. Both versions of GFDs are discussed here and some applications are given to straight lines and circles, which are an important class of curves.

2.3.1. Two definitions of GFDs

In order to allow also non-clockwise oriented, non-simple, open curves, a parameter ξ is introduced. The first definition of GFDs starts with the following transformation of the phase function $\phi(s)$:

$$\phi^* = \phi(\frac{Lt}{2\pi}) + \frac{\xi t}{2}, \quad with\ \xi = \frac{\phi(L)}{-\pi} \qquad \text{(2-10)}$$

It can be seen that this definition is compatible with non-simple, non-clockwise oriented open curves. If $\xi = 2$, the definition for FDs is obtained. Since $\xi = \phi(L)/-\pi$, it follows that $\phi(L) = -\xi\pi$ which is, in the $\xi = 2$-case, equal to -2π, thus describing a closed curve, according to the definition of FDs. If $\xi \neq 2$, the curve γ is an open curve. The distance parameter t ranges from 0 to 2π. Thus, by introducing the parameter ξ, the Fourier Descriptors-concept is generalised for more common use. The remaining transform considerations are the same as for the FDs, resulting in the same expressions for the Fourier coefficients a_k and b_k. The expression for μ_0 is slightly different with the introduction of the parameter ξ :

$$\mu_0 = -\frac{\pi\xi}{2} - \frac{1}{L}\sum_{k=1}^{m} l_k \Delta\phi_k \qquad (2\text{-}11)$$

Vieveen and Prasad [4] also give a perturbation analysis, that is, expressions for the Fourier coefficients of perturbated curves.

A second definition of GFDs was introduced by Weyland and Prasad [7]. In this definition, the same parameters are used as in the previous definition, but the ranges of the parameters are modified. In this case the transform in s-domain is as follows :

$$\phi^*(t) = \phi(Lt) + 2\pi\xi t \qquad (2\text{-}12)$$

The parameters t and ξ are defined as :

$$t \triangleq \frac{s}{L} \qquad (2\text{-}13)$$

$$\xi \triangleq \frac{-\phi(L)}{2\pi} \qquad (2\text{-}14)$$

where $0 \leq t \leq 1$, and $\xi = 1$ for a simple, clockwise oriented, closed curve. The spectral consideration remain the same as in the FDs-case, where this definition for $\phi^*(t)$ should be filled in. In this new definition, the distance parameter t is more naturally normalised to [0, 1] instead of [0, 2π]. Also ξ has a more natural normalisation : for $\xi = 1$ we have a closed curve.

2.3.2. GFDs applied on two important curves

In this section we will discuss the application of GFDs according to Weyland and Prasad (eq. (2-12 to (2-14)), to the special case of a linear function $\phi^*(t)$. It is useful to point out here, that this case covers two cases of the curve γ.

First, when $\phi(s)$ is constant, γ is a straight line. When transforming the angular function $\phi(s)$ to the normalised angular function $\phi^*(t)$, the latter will also be a constant function, since $\xi = \phi(L) = 0$. It is important to notice however, that when the curve ϕ under consideration is part of a curve with starting angle different from final angle (which implies that $\xi \neq 0$), the constant part will be increased by a linear part equal to $2\pi\xi t$. So in this case (which is a common case) the normalised phase function $\phi^*(t)$ is not a constant, yet it represents a straight line section.

Secondly, when $\phi(s)$ is a linear function, γ is a circle, since circles are the only figures that possess constant differential angles, measured along the arc distance. In this case, the $\phi^*(t)$-function will be the sum of two linear functions (one from the original linear $\phi(s)$-function and one linear addition from the fact that $\xi \neq 0$), which is generally a linear function itself, or a constant in exceptional cases.

Now that we have pointed out that a linear function $\phi^*(t)$ is a powerful function to examine, we will calculate the GFDs for this case.

Suppose that $\phi^*(t)$ is a linear function which connects the points $\{t_1, \phi_1\}$ and $\{t_2, \phi_2\}$, as plotted in figure 2.

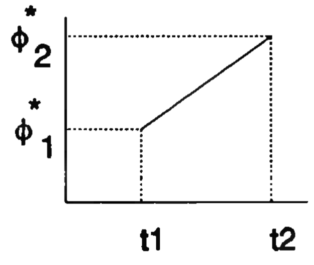


Figure 2 Linear normalised phase function $\phi^*(t)$

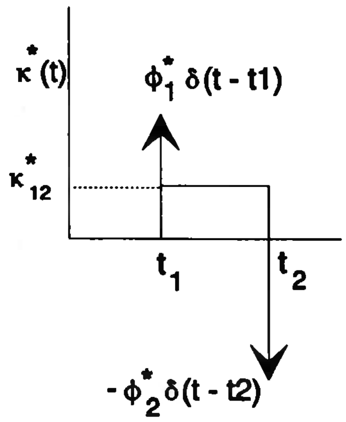


Figure 3 Derivative $\kappa^*(t)$ of $\phi^*(t)$

We have included also the derivative of $\phi^*(t)$, $\kappa^*(t)$ in fig. 3, because it will appear to be easier for calculations. As κ^* is defined as :

$$\kappa^* \triangleq \frac{d\phi^*(t)}{dt} \tag{2-15}$$

we have the following expression for $\kappa^*(t)$:

$$\kappa^*(t) = \kappa_{12}^*\{U(t-t_1) - U(t-t_2)\} + \phi_1^*\delta(t-t_1) - \phi_2^*\delta(t-t_2) \tag{2-16}$$

where U(t) denotes the unit step function and $\kappa^*_{12} = (\phi_2^* - \phi_1^*)/(t_2 - t_1)$. Because $\kappa^*(t)$ is the derivative of $\phi^*(t)$, the following relation holds in frequency-domain :

$$\Phi^*(f) = \frac{1}{j2\pi f} K^*(f) + \frac{1}{2} K^*(0)\delta(f) \tag{2-17}$$

We observe here that

$$K(0) = \int_{-\infty}^{\infty} \kappa^*(t)\, dt = (\phi_2 - \phi_1) + \phi_1 - \phi_2 = 0 \tag{2-18}$$

When working out these formulae we obtain the following results for the real and imaginary part of the frequency spectrum $\Phi^*(t)$:

$$Re[\Phi^*(f)] = \frac{\kappa_{12}^*}{4\pi^2 f^2}(\cos 2\pi f t_2 - \cos 2\pi f t_1) + \frac{\phi_2^*}{2\pi f}\sin 2\pi f t_4 - \frac{\phi_1^*}{2\pi f}\sin 2\pi f t_1 \tag{2-19a}$$

and

$$Im[\Phi^*(f)] = \frac{-\kappa_{12}^*}{4\pi^2 f^2}(\sin 2\pi f t_2 - \sin 2\pi f t_1) + \frac{\phi_2^*}{2\pi f}\cos 2\pi f t_2 - \frac{\phi_1^*}{2\pi f}\cos 2\pi f t_1 \tag{2-19b}$$

Furthermore, it can be shown that, if $\phi^*(t)$ is a real function (which is the case), then the Fourier coefficients are related to the frequency spectrum $\Phi^*(f)$ as follows :

$$\mu_0 = f_0\Phi^*(0) \tag{2-20a}$$

$$a_k = 2f_0 \, Re[\Phi^*(kf_0)] \tag{2-20b}$$

$$b_k = -2f_0 \, Im \, [\Phi^*(kf_0)] \tag{2-20c}$$

When filling in the expressions for the real and imaginary part of the spectrum, we obtain the following expressions for the Fourier coefficients :

$$\begin{aligned} a_k = {} & \frac{\phi_2^* - \phi_1^*}{2\pi^2 k^2 f_0 (t_2 - t_1)} (\cos 2\pi k f_0 t_2 - \cos 2\pi k f_0 t_1) \\ & + \frac{\phi_2^*}{\pi k} \sin 2\pi k f_0 t_2 - \frac{\phi_1^*}{\pi k} \sin 2\pi k f_0 t_1 \end{aligned} \tag{2-21a}$$

$$\begin{aligned} b_k = {} & \frac{\phi_2^* - \phi_1^*}{2\pi^2 k^2 f_0 (t_2 - t_1)} (\sin 2\pi k f_0 t_2 - \sin 2\pi k f_0 t_1) \\ & + \frac{\phi_2^*}{\pi k} \cos 2\pi k f_0 t_2 - \frac{\phi_1^*}{\pi k} \cos 2\pi k f_0 t_1 \end{aligned} \tag{2-21b}$$

The GFDs are obtained by substituting these results into eq. (2-7).

Summarising, we now have formulae for calculating the GFDs for the important class of straight lines and circles. At this point, it is obvious that many handwritten characters either consist of these two basic curves, or can be approximated by these. In succeeding chapters this idea will be developed and the formulae obtained above, will be applied on actual handwritten characters.

But first we have to take some resolution aspects into consideration, since they pose constraints on the number of samples and the frequency resolution.

2.3.3. Resolution requirements for GFDs in time and frequency domains.

Since we use a writing tablet or digitizer for recording curve-data, a sampling process is involved. It can intuitively be felt, that certain curves cannot be detected by the digitizer, if the resolution is too small. In this respect, we will speak of the spatial bandwidth W of the curve γ. This bandwidth has, analoguous to time signals, a relation with the curvature function $\kappa^*(t)$, as follows :

$$W \triangleq \frac{1}{2\pi} \, |\kappa^*(t)|_{\max} \qquad (2\text{-}22)$$

where $\kappa^*(t) = d\phi^*(t)/dt$. According to the Nyquist sampling theorem, $f_s = N/T \geq 2W$. Furthermore, $T \geq 1$, because t runs from 0 to 1. It follows then that

$$|\kappa^*(t)|_{\max} \leq \frac{\pi N}{T} \qquad (2\text{-}23)$$

or,with the definition of $\kappa^*(t)$:

$$W \leq \frac{N}{2T} \leq \frac{N}{2} \qquad (2\text{-}24)$$

At this point, we can decide to assign optimal resolution. By definition, optimal resolution is achieved, when the resolution in the t-(spatial) domain equals the resolution in the f-(frequency) domain. To be more precise, equal resolution means that the number of nonzero samples are (nearly) equal in both domains. Since $0 \leq t \leq 1$, and $\phi^*(t)$ does not contain frequency components larger then W, the maximum number of nonzero t-samples is then

$$N_t = \frac{1}{T_s} = \frac{N}{T} \qquad (2\text{-}25)$$

with $T_s = T/N$ the sampling time. The number of nonzero frequency-samples is :

$$N_f = \frac{W}{f_0} = WT \qquad (2\text{-}26)$$

Optimal resolution is, by definition, obtained if $N_t = N_f$, leading to :

$$T = \sqrt{\frac{N}{W}} \qquad (2\text{-}27)$$

When we substitute this value into the resolution terms, the following maximum number of nonzero samples will be :

$$N_t = N_f = \sqrt{NW} \tag{2-28}$$

Thus, when we know the maximal curvature $\kappa^*(t)$ (and thus W), and the total number of samples N, we have defined the maximum number of nonzero samples for both t- and f-domain, which can be considered optimal scaling conditions for both domains.

CHAPTER 3. CURVE-FITTING USING ANGULAR DATA

3.1. INTRODUCTION

The application of the formulae, discussed in chapter 2 requires curves that consist of straight line pieces and arcs of circles exclusively. Naturally, man's handwritten characters do seldom possess this property. Therefore, in order to use the formulae, we first have to process the data which comes from the writing tablet. Here we can make a choice whether to use the absolute angles or the ordinary x-y-coordinates. In this chapter angular data will be discussed. Also some drawbacks will be found when choosing for the angular data. These drawbacks will be discussed in the last paragraph of this chapter.

3.2. PARTITIONING THE SET OF ANGULAR DATA

Assume that we have a set of N absolute angles $\{\phi[1]...\phi[N]\}$ at our disposal. These angles come directly from the writing tablet. As an example fig.4 gives such a plot which consists of 16 samples.

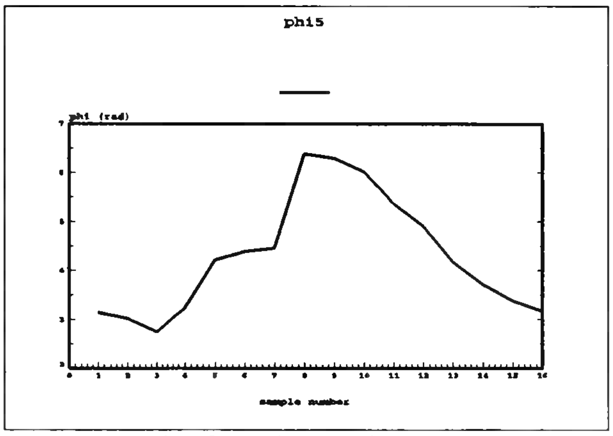


Figure 4 ϕ-plot for a handwritten pattern.

As expected, the plot does not consist of large pieces of straight lines. We now want to make the following partition :

$\{\phi[1]...\phi[N]\} = \{\{\phi[1]..\phi[k]\}, \{\phi[k+1]..\phi[m]\},..,\{\phi[p+1\}..\phi[N]\}\}$, where each subset consists of points of one straight line. This partition results in sections of straight lines, thus the corresponding curve in the x-y-plane consists of straight line pieces (when ϕ is constant) and arcs of circles (when ϕ is a linear function).

Next, a rule must be found to make the partition. A suitable function to use is the first derivative of ϕ, that is $\Delta\phi$ ($\Delta\phi = \phi_i - \phi_{i-1}$). When the absolute value of $\Delta\phi$ exceeds a given value of (π - ϵ), then the preceding points belong to the same subset and the following points belong to the next subset. All values of $\Delta\phi$ are checked likewise and finally we have the partitioning of all N points of ϕ.

As an illustration, the first derivative function of the function from fig.4 is here plotted in fig. 5.

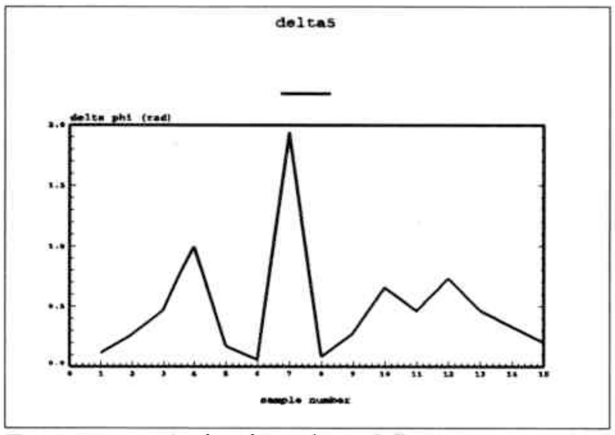


Figure 5 $\Delta\phi$-derivative plot of fig.4

We can indicate the two biggest peaks in fig. 5 at sample numbers 4 and 7. Therefore, we can make the following partitioning: $\{\phi[1..N]\} = \{\{\phi[1]..\phi[4]\}, \{\phi[5]..\phi[7]\}, \{\phi[8]..\phi[16]\}\}$.

The next step is to obtain the best straight line pieces for each sub set. This is will be discussed in the next section.

3.3. CURVE-FITTING TECHNIQUES TO APPROXIMATE ARBITRARY CURVES WITH STRAIGHT LINES AND CIRCLES

To obtain straight lines from a data set we use the least squares technique [9]. This well known technique gives the best solution in the sense of least squares. This section will treat calculation for one subset of angles ϕ only, since the calculation is the same for all subsets.

Suppose that we have a subset $\{\phi[k+1]..\phi[k+n]\}$, consisting of n points. These n points have to be fitted in the best linear function of the form $\phi_{ls}[i] = C\cdot i + D$,where i=k+1,..,k+n, and C and D unknown, real numbers. We can write this as A$\underline{x}$ = $\underline{b}$ which is the following equation :

$$\begin{bmatrix} 1 & k+1 \\ 1 & k+2 \\ 1 & k+3 \\ . & . \\ . & . \\ 1 & k+n \end{bmatrix} \bullet \begin{bmatrix} D \\ C \end{bmatrix} = \begin{bmatrix} \phi[k+1] \\ \phi[k+2] \\ \phi[k+3] \\ . \\ . \\ \phi[k+n] \end{bmatrix} \tag{3-1}$$

This equation has no solution for $[D \quad C]^T$ (if the columns are independent) since the system is overdetermined. However, we can find the best solution in the sense of least squares, that is minimizing the sum of errors :

$$e = \sum_{i=1}^{n} (\phi_{ls}[i] - \phi[i])^2 \tag{3-2}$$

The solution which minimizes this error sum is given in Strang [9] :
$\underline{x}_{ls} = (A^TA)^{-1}A^T\underline{b}$, which can in our case be filled in as :

$$\frac{1}{n\sum_{i=1}^{n}(k+i)^2 - \left[\sum_{i=1}^{n}(k+i)\right]^2} \cdot \begin{bmatrix} \sum_{i=1}^{n}(k+i)^2 & -\sum_{i=1}^{n}(k+i) \\ -\sum_{i=1}^{n}(k+i) & n \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & . & . & 1 \\ k+1 & k+2 & . & . & k+n \end{bmatrix} \cdot \begin{bmatrix} \phi[k+1] \\ \phi[k+2] \\ . \\ . \\ \phi[k+n] \end{bmatrix} \tag{3-3}$$

Working out these equations result in expressions for C and D:

$$D = \frac{(\sum_{i=1}^{n} (k+i)^2).(\sum_{i=1}^{n} \phi[k+i]) \; - \; (\sum_{i=1}^{n} (k+i)\;).\;(\sum_{i=1}^{n} (k+i).\phi[k+i])}{n \; . \; \sum_{i=1}^{n} (k+i)^2 \; - \; (\;\sum_{i=1}^{n} (k+i)\;)^2} \qquad \text{(3-4a)}$$

$$C = \frac{-(\sum_{i=1}^{n} (k+1)).(\sum_{i=1}^{n} \phi[k+i]) \; + \; n.(\sum_{i=1}^{n} (k+i).\phi[k+i])}{n \; \sum_{i=1}^{n} (k+i)^2 \; - \; (\;\sum_{i=1}^{n} (k+i)\;)^2}$$

These expressions can now be filled in the equation $\phi_{ls}[i] = C \cdot i + D$, resulting in the best straight line according to the least squares error.

We could now apply these results to all sections of the ϕ-plot and we would then have a ϕ_{ls}-plot which would consist of straight line pieces only. However, we will not go further here because this method has some drawbacks which will be discussed in the next section.

3.4. DRAWBACKS WHEN EMPLOYING THE ANGLES AS INPUT DATA

As mentioned, the method described in the previous section has some drawbacks, that is why we will not use it in the course of the investigation.

First, the angles give only information about the successive angles in the drawn curve. Equal sampling distance along the arc of the curve is assumed. However, when employing DCC-codes, not all angles correspond with the same arc length. This can be seen from fig. 6 on the next page : the distance from the origin to point 1 is $\sqrt{2}$ times bigger than the distance to point 2.

Secondly, when applying the least squares technique to the ϕ-plot, we do not consider the consequences in the x-y-domain. We will be ensured that there will be only straight lines and arcs of circles in the new curve, but we do not know to what extend the curve has changed in the x-y-plane.

When we use the x-y-coordinates of the curve instead, we have none of the disadvantages mentioned before. We can then compute the best straight lines in the x-y-plane. But we must in this case also compute the best circles in the x-y-plane. Fortunately, numerical methods exist to obtain these. Employing x-y-coordinates has the advantage that we can clearly define an error criterion in the x-y-plane which has to be minimized in order to approximate for the best curve

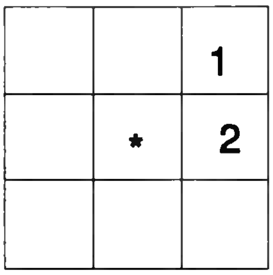


Figure 6 DCC frame : fields 1 and 2 have different distances to the centre.

that consists of straight line pieces and arcs of circles only. In the next chapter we will discuss this method.

CHAPTER 4. CURVE-FITTING IN THE X-Y-PLANE

4.1. INTRODUCTION

In the previous chapter a method was developed for curve-fitting using angular data. However, some drawbacks were found as well. In this chapter we will discuss curve-fitting techniques in the X-Y-plane. The disadvantages as encountered in the previous chapter will vanish, and important advantages will appear. We will try to find the best estimate in some sense for the written curve. The best straight lines in the x-y-plane have to be found as well as the best circles. Also a suitable rule must be found to decide whether to apply the straight line approximation or the circle approximation. All these matters will be covered in this chapter.

First, straight-line approximation will be discussed and then circular approximation.

4.2. FITTING STRAIGHT LINES TO AN ARBITRARY SET OF X-Y-POINTS.

When we want to find the 'best' straight lines, given a set of x-y-coordinate pairs, we first have to define what is best. Therefore we will consider a suitable error criterion in the next subsection. Once we have defined this criterion we can optimize the error with respect to this criterion.

4.2.1. Error criterion for straight line fitting.

At this point we clearly have to distinguish from chapter 3, in which we have defined the error to be minimized as stated in formula (3-2). There, we assume uncertainty to occur in the ϕ-data only, not in the i-values (sample numbers). Therefore, we have used the least squares technique.

When working with x-y-coordinates however, uncertainty occurs in the x-coordinate as well as in the y-coordinate. We may no longer use the least squares technique. Instead, we will use the so called total least squares technique (TLS).

To give a better understanding of the error criterion we have shown two plots in figures 7 and 8. In fig. 7 the least squares are applied, while in fig. 8 the total least squares are applied.

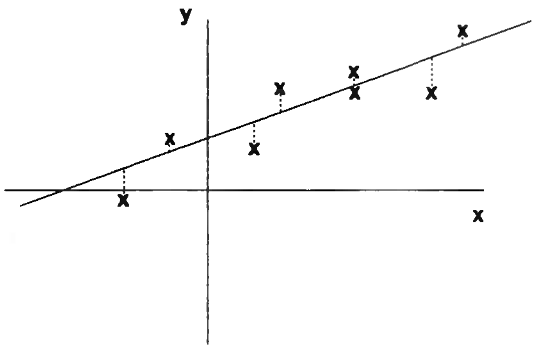


Figure 7 Least squares errors

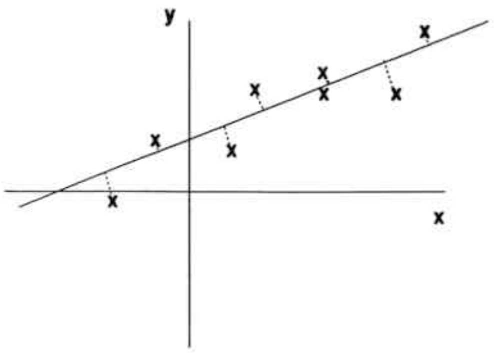


Figure 8 Total least squares error

We can also formulate the error considered in the total least squares problem. We give some equations of importance here. First we want a linear function

$$y_i = A\,x_i + B \tag{4-1}$$

This can be written as the following algebraic equation :

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ . & . \\ . & . \\ x_n & 1 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix} \tag{4-2}$$

which has the form C$\underline{z}$ = $\underline{y}$, where $\underline{z}$ is the unknown vector $[A \;\; B]^T$ and uncertainties occurring in the matrix C (which contains the x-coordinates) and the vector $\underline{y}$. If we include the error terms in the formula we have : $(C + E)\underline{z} = \underline{y} + \underline{r}$, where E is the error matrix due to uncertainties in the x-coordinates and $\underline{r}$ is the error vector due to uncertainties in the y-coordinates. Now the total least error problem can be formulated ([11], [14]):

$$\text{minimize } \|[E|\underline{r}]\|_F \tag{4-3}$$

where $[E|\underline{r}]$ is the matrix obtained after putting the vector $\underline{r}$ behind the matrix E. Furthermore, $\| \;\|$ denotes the Frobenius norm, e.g. $\|P\|_F^2 = \Sigma_j\Sigma_i\,|p_{ij}|^2$. (For a comparison, minimizing $\|(C + E)\underline{z} - (\underline{y} + \underline{r})\|_2$ would solve the least squares problem). When a minimizing matrix $[E|\underline{r}]$ is found, then any vector $\underline{z}$ satisfying eq.(4-3) is said to solve the total least squares problem.

4.2.2. Solving the Total Least Squares problem.

When solving the total least squares problem (that is minimizing the total error matrix as mentioned, we need the singular value decomposition of the matrix $[C|\underline{y}]$, that is the data matrix which consists of the matrix C and the vector $\underline{y}$.([11]). This gives a factorization of an mxn-matrix P into $P = U\Sigma V^T$, where Σ is an nxn- diagonal matrix which contains the singular values of the matrix P, and U and V are orthogonal matrices. A discussion of this decomposition and its computation is given in appendix A.

Once we have obtained the singular value decomposition of the data-matrix $[C|\underline{y}]$, we have to normalize the column of the matrix V corresponding with the smallest singular value. This vector will be the solution $\underline{z} = [A \ \ B]^T$ to the total least squares problem. Assume that we have the singular value decomposition $C = U\Sigma V^T$, with $U = \{u_1,...u_{n+1}\}$ and $V = \{v_1,...,v_{n+1}\}$ the column partitions of U and V. Note that there are n+1 columns because we consider the decomposition of the n column matrix C extended with the vector $\underline{y}$ which makes the matrix $[C|\underline{y}]$ (n+1)-columned. Let the singular values σ be ordered as follows : $\sigma_1 \geq ... \geq \sigma_k > \sigma_{k+1} = ... = \sigma_{n+1}$. Let $S_C = \text{span}\{v_{k+1},...,v_{n+1}\}$. Suppose we can find a vector v in S_C having the form

$$\underline{v} = \begin{bmatrix} \underline{y} \\ \alpha \end{bmatrix} \quad \underline{y} \in R^n \ , \quad \alpha \neq 0 \tag{4-4}$$

then the solution is then given by

$$\underline{z} = [A \ \ B]^T = -\underline{y}/\alpha. \tag{4-5}$$

If σ_{n+1} is a repeated singular value, then there is no unique solution. However, in this case we can find a unique solution by applying a suitable Householder transformation :

$$[\ v_{k+1},...,v_{n+1}\ ] \cdot Q = \begin{bmatrix} W & \underline{y} \\ 0 & \alpha \end{bmatrix} \begin{matrix} \{\, n\text{-}k \\ \{\, 1 \end{matrix} \tag{4-6}$$

4.2.3. Some calculation examples to obtain Total Least Squares solutions.

Here we will bring the theory of the Singular Value Decomposition into practice. We will first define a set of (x,y) points on which calculations will be performed. Suppose that we have the following set of data points

Table I : Pairs of X-Y-coordinates to use for total least squares problem.

x	1	2	2	2	3	4	4	4	5	6
y	0	0	1	2	2	2	3	4	4	4

These coordinates can be plotted out in graphics and are displayed in figure 9.

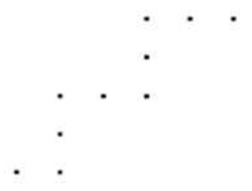


Figure 9 X-Y-coordinates for total least problem

The problem now is to obtain the best straight line solutions in y = ax + b. This system can be written as y = [x, [1 1 .....1 $]^T$] . [a b $]^T$, for our set of data leading to :

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 2 \\ 2 \\ 3 \\ 4 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 2 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 4 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} \qquad (4\text{-}7)$$

For purpose of Least Squares Solutions, as described earlier, we have to rearrange this system in the following way : $[\,\mathbf{x}, [\,1\ 1\ 1 \ldots\ldots 1\,]^T, \mathbf{y}\,] = \mathbf{0}$. The singular value decomposition for this matrix is $A' = U\,\Sigma\,V^T$ and it is as follows :

$$\begin{bmatrix} 1.000 & 1.000 & 0.000 \\ 2.000 & 1.000 & 0.000 \\ 2.000 & 1.000 & 1.000 \\ 2.000 & 1.000 & 2.000 \\ 3.000 & 1.000 & 2.000 \\ 4.000 & 1.000 & 2.000 \\ 4.000 & 1.000 & 3.000 \\ 4.000 & 1.000 & 4.000 \\ 5.000 & 1.000 & 4.000 \\ 6.000 & 1.000 & 4.000 \end{bmatrix} = \qquad (4\text{-}8)$$

$$\begin{bmatrix} -0.069 & 0.436 & 0.389 \\ -0.125 & 0.637 & -0.024 \\ -0.165 & 0.276 & 0.271 \\ -0.205 & -0.084 & 0.567 \\ -0.260 & 0.117 & 0.154 \\ -0.316 & 0.317 & -0.259 \\ -0.356 & -0.043 & 0.036 \\ -0.396 & -0.404 & 0.332 \\ -0.451 & -0.203 & -0.081 \\ -0.507 & -0.003 & -0.494 \end{bmatrix} \cdot \begin{bmatrix} 14.334 & 0.000 & 0.000 \\ 0.000 & 2.105 & 0.000 \\ 0.000 & 0.000 & 1.053 \end{bmatrix} \cdot \begin{bmatrix} -0.795 & -0.199 & -0.573 \\ 0.422 & 0.497 & -0.758 \\ -0.435 & 0.845 & 0.311 \end{bmatrix}$$

with the diagonal matrix containing the three different singular values. Note that the matrices U and V^T are orthogonal. Also note that the singular values are put in decreasing order. In general the system is overdetermined, that is, the singular values are different. In this case the solution for the vector [a b $]^T$ is given by [-v[3,1]/v[3,3], -v[3,2]/v[3,3]] = [0.435/0.311 , -0.845/0.311] = [1.39 , -2.71]. This means that the equation is : y = 1.39x - 2.71. This function is plotted together with the sample points in figure 10.

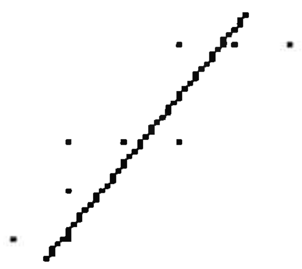


Figure 10 Total least squares solution yielding a straight line

4.3. FITTING ARCS OF CIRCLES TO AN ARBITRARY SET OF X-Y-POINTS.

We will now try to find the best circles, given a set of x-y-coordinates. As in the previous sections, we also have to define an error criterion which is to be minimized. In this case we will have to cope with circles which are by nature non-linear. Therefore we cannot make use of the algebraic convenience which we had in the previous sections. Instead, we now have to make use of numerical methods to find the best arcs of circles. Fortunately, a suitable method is found and will be discussed.

4.3.1. An error criterion for circle-fitting.

In figure 11 on the next page the errors are plotted as the dark lines. These errors are similar as in the straight line case, but the formula for the error will be different since we are dealing with circles now. Figure 11 gives an intuitive justification of the error criterion.

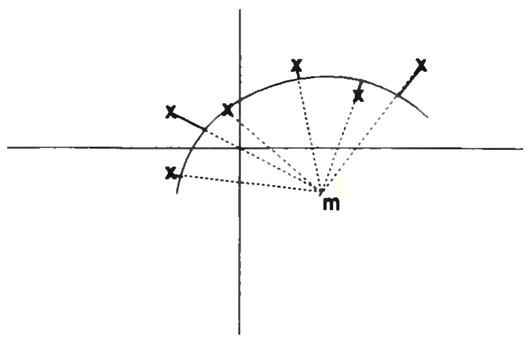


Figure 11 Errors in estimation of circle arc

In order to perform calculations we have to cast the error in a formula. We observe that

$$e = \sum_{i=1}^{n} \left((x_i - m_x)^2 + (y_i - m_y)^2 \quad - \quad r^2\right)^2 \tag{4-9}$$

with m_x the x-coordinate of the centre of the circle, m_y the y-coordinate of the centre, and r the radius of the circle. So we have three parameters, to which we will optimize. The error is just the square of the difference between the Euclidian distances from the data points to the centre of the circle, and the squared radius of the circle, summed over all the data points.

When we work out this formula we get :

$$\begin{aligned} e = \sum_{i=1}^{n} \, [\, & x_i^4 - 4m_x x_i^3 - 4m_x^3 x_i + 6x_i^2 m_x^2 + m_x^4 + y_i^4 - 4m_y y_i^3 - 4m_y^3 y_i + 6y_i^2 m_y^2 + \\ & m_y^4 + 2x_i^2 y_i^2 - 4x_i y_i^2 m_x + 2y_i^2 m_x^2 - 4x_i^2 y_i m_y + 8m_x m_y x_i y_i - \\ & 4m_x^2 m_y y_i + 2m_y^2 x_i^2 - 4m_x m_y^2 x_i + 2m_x^2 m_y^2 - 2r^2 x_i^2 + 4r^2 x_i m_x - \\ & 2r^2 m_x^2 - 2r^2 y_i^2 + 4r^2 y_i m_y - 2r^2 m_y^2 + r^4 \;] \end{aligned} \tag{4-10}$$

This error has to be minimized with respect to the three circle parameters m_x, m_y and r. That means that we have to take the first derivatives with respect to m_x, m_y and r and set them equal to 0. The resulting non-linear system is not easily solved. For this reason we have to solve this problem numerically. Several methods exist for solving these non-linear problems [10]. First a motivation for use of the simplex-method will be given.

4.3.2. THE SIMPLEX-METHOD TO SOLVE THE NON-LINEAR TOTAL LEAST SQUARES PROBLEM

4.3.2.1. Motivating the use of SIMPLEX-method

We first want to give a motivation for choosing the simplex method, before starting to work with it. It is clear that we are forced to use some numerical method, since the expression in eq. (4-10) is too complex to obtain a direct analytical solution.

Various numerical methods for optimizing a function exist (see [10]), often using the derivative of the function to be optimised (steepest descent methods, among others.). These methods imply that the derivative must be calculated first. This calculation would in our case be executed three times, since we have a three-parametered error function $\epsilon(m_x, m_y, r)$. Considering the complexity of eq. (4-10), it would be wise to use a method which does not need these calculations, since they can easily introduce writing mistakes. The simplex-method does not require these derivatives, it only requires first estimates of the parameters (m_x, m_y, r).

Furthermore, since the X-Y-coordinates of handwritten curves are generally not functions (where one X-coordinate must correspond at the most to one Y-coordinate), but arbitrary curves. It then seems preferable not to use gradient methods. Moreover, the simplex-method is not difficult to understand graphically, in contrast with the analytical gradient methods. Given these considerations, we will now discuss and apply the simplex-method. In order to avoid confusion, we stress here that the simplex-method, used here, is not the more famous linear programming simplex-method. The simplex-method which we will use, is a non-linear optimising algorithm.

4.3.2.2. Initial estimates for (m_x, m_y, r).

When we have N points under consideration, we can start the estimation by observing that the centre of the circle roughly lies on the perpendicular bisector of the line connecting the first point (x_1, y_1) and the last point (x_N, y_N). The situation is depicted in fig. 12 on the next page.

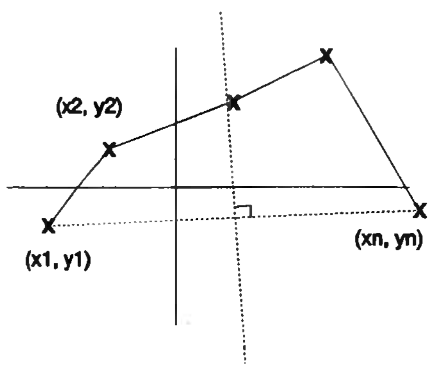


Figure 12 Some data points and perpendicular bisector for estimation of centre of the circle.

We can easily write the vector representation of this line. Since the direction of the line piece between (x_1, y_1) and (x_N, y_N) is given by

$$\begin{bmatrix} x \\ y \end{bmatrix} = \lambda \cdot \begin{bmatrix} y_{x_N} - x_1 \\ y_N - y_1 \end{bmatrix} \quad , \quad with \lambda \in \mathbf{R} \tag{4-11}$$

and the midpoint of the bisection is given by

$$\begin{bmatrix} x_{mp} \\ y_{mp} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(x_1 + x_N) \\ \frac{1}{2}(y_1 + y_N) \end{bmatrix} \tag{4-12}$$

the vector representation of the perpendicular bisector is

$$\begin{bmatrix} m_x \\ m_y \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(x_i + x_N) \\ \frac{1}{2}(y_1 + y_N) \end{bmatrix} + \lambda \begin{bmatrix} (y_N - y_1) \\ -(x_N - x_1) \end{bmatrix} \tag{4-13}$$

Now we have to find a suitable value for λ. This value has a certain relation with the radius of the circle. If the estimation of the radius r_{estim} is known, we can make the following choice for λ :

$$\lambda_{estim} = \pm \frac{r_{estim}}{\sqrt{(y_N - y_1)^2 + (x_N - x_1)^2}} \tag{4-14}$$

which means that λ has the value of the estimated radius, normalized to the length of the direction vector.

Now we only have to make a proper estimation for the radius r. We base this estimation on the knowledge that the curvature of a circle, $\kappa = d\theta/ds$, where θ is the angle along the arc length s. We can then take the mean value of κ :

$$\kappa_{mean} = \frac{\displaystyle\sum_{i=3}^{n} \frac{\arctan\left(\dfrac{y_i - y_{i-1}}{x_i - x_{i-1}}\right) - \arctan\left(\dfrac{y_{i-1} - y_{i-2}}{x_{i-1} - x_{i-2}}\right)}{\sqrt{(x_i - x_{i-2})^2 + (y_i - y_{i-2})^2}}}{(N-2)} \tag{4-15}$$

Note that the mean is taken over the (N-2) available angle difference-points, since N points correspond with (N-2) difference angles. Now $r_{estim} = 1/\kappa$ and we have found an estimate for the radius. This can now be filled into equation to obtain the corresponding centre (m_x, m_y).

This first estimate can now easily be used to obtain the other three vertices of the simplex. We could for example take the following four points : $\{(m_{x,\,estim}, m_{y,\,estim}, r_{estim}), (m_{x,\,estim}+\delta, m_{y,\,estim}, r_{estim}+\delta), (m_{x,\,estim}, m_{y,\,estim}+\delta, r_{estim}), (m_{x,\,estim}, m_{y,\,estim}, r_{estim}+\delta)\}$ where δ is a small number tat could for example be set equal to $r_{estim}/10$.

Now we have completed our task to find four starting points . These points will be submitted to the simplex method, which is an iterative method, converging to the minimal error solution. It will be discussed in the next section.

4.3.2.3. Converging to the minimal error

Starting with the first simplex, the coordinates are filled in the error function (eq. 4-10) for each of the four vertices. The highest value is taken out and reflected. this reflection process is illustrated in fig. 13 on the next page.

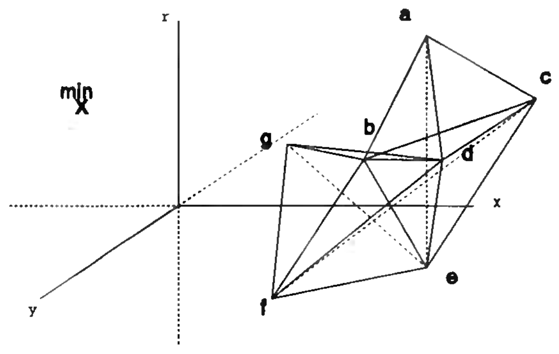


Figure 13 Reflection process in the simplex method

As shown in fig. 13, the points a,b,c and d span the initial simplex. The coordinates of point a give the largest value when filled in the error criterion. This point a is then reflected to point e and the a new simplex {b,c,d,e} is formed. from this simplex, point c gives the largest error, so it is reflected to point f, resulting in the new simplex {b,d,e,f}. This process is repeated until the error differences in proceeding points are small enough. A suitable convergence criterion is [10] :

$$Q = \sqrt{\sum_{i=1}^{n} \frac{[e(m_{x,i},m_{y,i},r_i) \;-\; e(m_{x,0},m_{y,0},r_0)]^2}{N+1}} \leq \epsilon \tag{4-16}$$

where the index i stands for the i-the vertex of the simplex, and the index 0 for the centroid of the simplex, defined as :

$$\begin{bmatrix} m_{x,0} \\ m_{y,0} \\ r_0 \end{bmatrix} = \frac{1}{n} \sum_{i=1,i\neq h}^{n+1} \begin{bmatrix} m_{x,i} \\ m_{y,i} \\ r_i \end{bmatrix} \tag{4-17}$$

where h stands for the index corresponding with the highest value of the error. The centroid is the point in the mirror plane point through reflection is performed. Now the reflection point can be found as :

where α is the reflection coefficient (can be set to 1), 0 means centroid and h means the highest error of the simplex points.

In fig. 13 it may occur that the after reflection of a to e,e is the point which results in the highest value of the error. This means that e is reflected to a and the process does not converge.

$$\begin{bmatrix} m_{x,reflex} \\ m_{y,reflex} \\ r_{reflex} \end{bmatrix} = (1+\alpha) \begin{bmatrix} m_{x,0} \\ m_{y,0} \\ r_0 \end{bmatrix} - \alpha \begin{bmatrix} m_{x,h} \\ m_{y,h} \\ r_h \end{bmatrix} \tag{4-18}$$

We can overcome this problem by making a rule that no return can be made to previous points.

If, after reflection, this new point is found corresponding to the smallest error in the simplex, we can assume that the minimum is in this direction. The process can be speeded up by expansion, that is, moving this point further in the direction of the centroid. The new point is then given by :

$$\begin{bmatrix} m_{x,e} \\ m_{y,e} \\ r_e \end{bmatrix} = \gamma \begin{bmatrix} m_{x,r} \\ m_{y,r} \\ r_r \end{bmatrix} + (1-\gamma) \begin{bmatrix} m_{x,0} \\ m_{y,0} \\ r_0 \end{bmatrix} \tag{4-19}$$

where γ is the expansion coefficient, r refers to the previously obtained reflection point, 0 to the centroid. The subscript e refers to the expanded point and will replace the reflected point, thus constructing a new simplex. After this operation a new reflection will be performed etc.

If, on the other hand, a point is found after reflection which corresponds to the largest value in the new simplex, we must make the simplex smaller because the minimum is somewhere in the simplex. This is accomplished by contraction :

$$\begin{bmatrix} m_{x,c} \\ m_{y,c} \\ r_c \end{bmatrix} = \beta \begin{bmatrix} m_{x,h} \\ m_{y,h} \\ r_h \end{bmatrix} + (1-\beta) \begin{bmatrix} m_{x,0} \\ m_{y,0} \\ r_0 \end{bmatrix} \tag{4-20}$$

where β is the contraction coefficient ($0 \leq \beta \leq 1$).

Rao [10] has given a flow chart of the simplex method. Based on this flow chart a Pascal program has been implemented, which is listed in appendix B. In the next section some illustrative calculations are performed, with the aid of the Pascal program.

4.3.3. Calculation results of circular fitting.

Some calculation experiments have been performed on circular matching as well. As an example, lets us consider the 10 pairs of x-y-coordinates listed in the following table.

Table II : Pairs of X-Y-coordinates to be used for solving the circular matching problem.

x	10	20	30	40	50	60	70	80	90	100
y	15	20	24	20	25	23	15	12	7	0

The plot that is represented by these coordinates is displayed in fig. 14. After computation according to the SIMPLEX-method, we obtain the following values for the circle parameters : m_x = 43.960, m_y = -48.138 and radius r = 71.635. In the calculations we have set the error criterion to smaller than 1E-5. Also a plot can be made together with the calculated circle, which is approximated according to the smallest error as defined in paragraph 3.3.1. The resulting plot is displayed in figure 14.

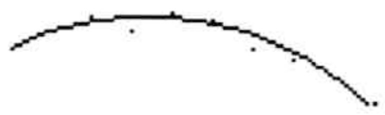


Figure 14 Circle matching, resulting in an arc of a circle

CHAPTER 5. APPLICATION OF X-Y-CURVE-FITTING ON HANDWRITTEN CHARACTERS.

5.1. INTRODUCTION.

So far we were able to extract the best approximations in the (total) least squares sense, for arbitrary sample data. We have applied the calculation methods on some ad hoc samples. In this chapter we will switch over to real applications, that is on handwritten characters. The written character first has to be divided into sections which can then be treated separately. First a partitioning process will be discussed, in order to divide the character into separate pieces. Then, some consideration will be given to the decision whether to assign circles or straight lines.

5.2. PARTITIONING THE CHARACTER.

5.2.1. Partitioning based on peak-detection.

For this purpose, the absolute difference of the relative, normalised angles, that is $|\Delta\phi^*[i]|$, will be used. It is felt, that whenever this function has a large value, partitioning must take place, on the other hand, when the absolute value of the difference in succeeding angles is not too large, these values will probably belong to a set of points that can be approximated either by a straight line or a circle arc. In this investigation, we have decided to allow for a maximum of 5 peaks in the character. Practice has shown that most characters of the alphabet can be split up into 3 to 5 sections. For example, the letter 'k', is a rather peaky character and contains 4 to 5 peaks in the absolute differential angle function $|\Delta\phi^*[i]|$, depending on the writing style. For use on other then Greek alphabets, the number of sections may be chosen higher.

As an example, the character 'k', was drawn on the writing tablet. The X-Y-coordinates were registered, containing 54 samples. From this data-file, a plot was draw (fig. 15 on the next page), displaying the character itself.

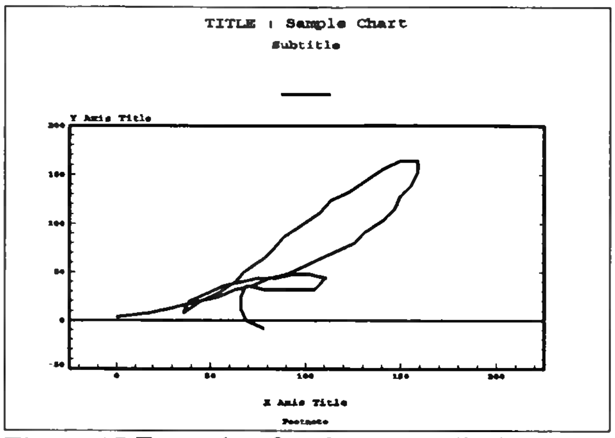


Figure 15 Example of a character, displayed on the basis of X-Y-coordinates.

From the X-Y-coordinates of this plot, the absolute angles $\theta[i]$ can be achieved, being $\theta[i] = \arctan((y_{i+1} - y_i)/(x_{i+1} - x_i))$. From these, the relative angles $\phi[i]$ can be obtained, from the relation $\phi[i] = \theta[i] - \theta[0]$. These points are displayed in fig. 16.

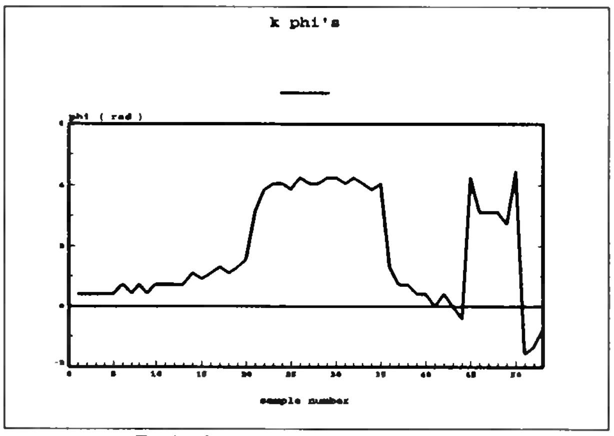


Figure 16 Relative angles $\phi[i]$ for a handwritten curve

For partitioning, we now wish to define large steps in the $\phi[i]$-plot in fig. 16. It is not quite obvious where these steps are from fig. 16. Therefore we will apply the difference operator on these data. We will calculate $|\Delta\phi[i]| = \text{abs}\{\phi[i+1] - \phi[i]\}$. The resulting plot is displayed in fig. 17 on the next page and it shows very clearly the peaks in difference at positions i = 21, 36, 45 an 51. From this plot, we can decide to make the partitioning $\{\phi[1]..\phi[54]\} = \{\phi[1]...\phi[20]\}$, $\phi[22]...\phi[35]\}$, $\{\phi[37]...\phi[44]\}$, $\{\phi[46]...\phi[50]\}$, $\{\phi[52]...\phi[54]\}$. Note that in this partitioning, we have not filled one sample between two successive subsets. In a following paragraph we will give attention to the filling in of these sections. Now we will discuss a peak-detection algorithm.

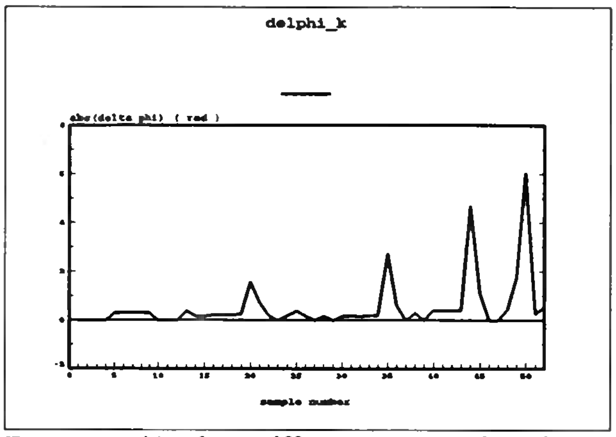


Figure 17 Absolute difference phase function to detect peaks.

5.2.2. A special peak-detection algorithm.

An algorithm has been implemented, which detects the five biggest peaks out of a set of N samples $[|\Delta\phi|_1,\ldots,|\Delta\phi|_N]$. Five peaks are enough, since characters have often only few peaks in their $|\Delta\phi|$ -function. Initially, the five first samples of the set $|\Delta\phi|_1,\ldots,|\Delta\phi|_5$ are ordered and stored in the variables peak[1]...peak[5], where peak[1] denotes the largest value and peak[5] the lowest. Also the positions where these peaks occur, are registered. Therefore, we introduce the variables peak_location[1],...,peak_location[5]. Peak_location[1] for example, denotes the position i where peak[1] (the largest peak) occurs. In fig. 18 the situation is outlined.

When the five first samples are ordered as described, all other remaining samples are examined. For example, if $|\Delta\phi|_6$ < peak[5], that is, smaller than the lowest peak, then this value $|\Delta\phi|_6$ will be neglected and the next sample (no.7) will be examined, etc. On the other hand, if $|\Delta\phi|_6$ > peak[5], several possibilities arise :

1) peak[4] > $|\Delta\phi|_6$ > peak[5]
   In this case, peak[5] at [peak_location[5] = 4, will be discarded and the new peak[5] will have the value of $|\Delta\phi|_6$ and peak_location[5] will become 6. This procedure is illustrated in fig. 19.

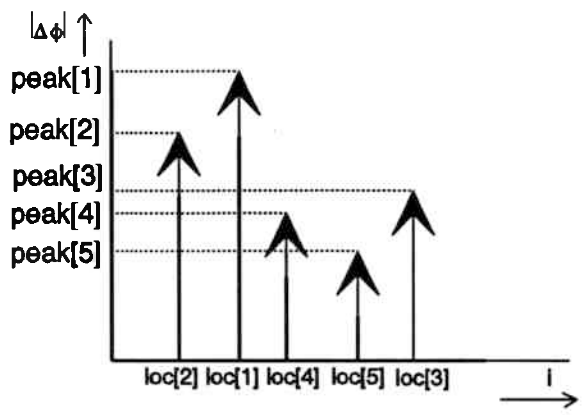


Figure 18 five peaks withe their ordered values and the positions where they are situated.

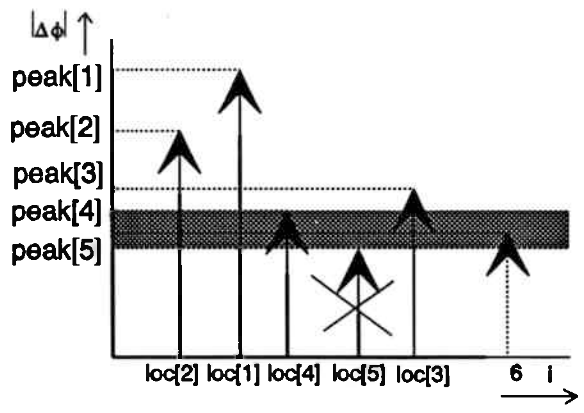


Figure 19 removal of an old peak when a new peak if found.

2) peak[3] > $|\Delta\phi|_6$ > peak[4].

Here, peak[5] at peak_location[5]=4, will be discarded. The new value of peak[5] will become the old value of peak[4] and peak_location[5] will become the old value of peak_location[4] (=3 in our case). Fig. 20 on the next page gives an illustration of this situation.

3) peak[2] > $|\Delta\phi|_6$ > peak[3]. This situation will be treated likewise.
4) peak[1] > $|\Delta\phi|_6$ > peak[2]. This situation will be treated likewise.
5) $|\Delta\phi|_6$ > peak[1]. This situation will be treated likewise.

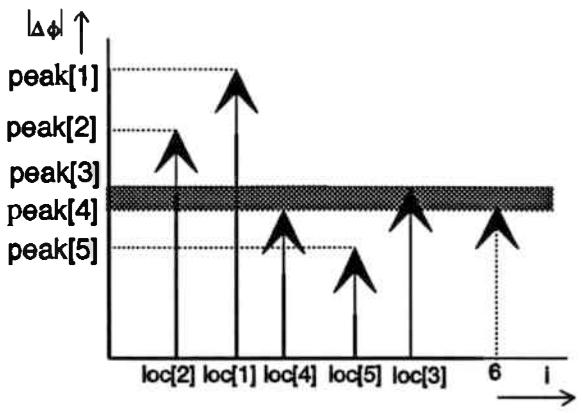


Figure 20 Another peak is found > peak[5].

So far, only sample number i = 6 was examined. After this point is examined as described, point number 7 will be examined in the same manner. This procedure continues until all N samples are examined. The resulting ordering gives the 5 biggest peaks of the N samples of $|\Delta\phi|$ -data.

5.2.3. Removal of 'false' peaks.

The peak-detection procedure has been explained in the previous paragraph, but the point was to extract 5 peaks from the set of N values $|\Delta\phi|$. In normal characters, there are often less than 5 peaks, which means that we have included too many peaks. In this section we will discuss the removal of these 'false' peaks. 'False' peaks can occur due to two reasons.

First, suppose that we have a peak, which is smeared out over two samples, as illustrated in fig. 21.

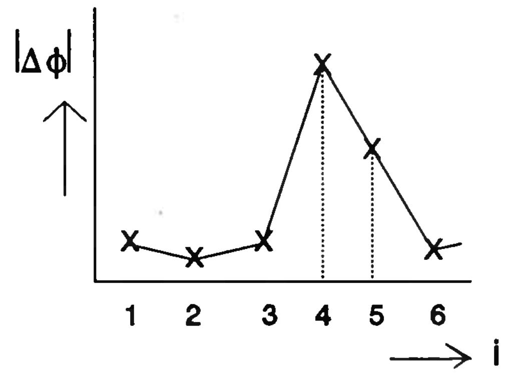


Figure 21 Two apparent peaks

Here, we would detect two peaks. at positions 4 and 5, while there is actual only one peak. This problem can be overcome by introducing a variable called DIST (from distance). If a new peak is found, (say, at location 5 in fig.18), it will only be considered a peak, if its location lies a distance > DIST away from the latest peak so far (that is the peak at location 4 in fig. 21). If distance > DIST, there is no danger that the peak is part of a bigger peak. When the distance of the current peak is smaller than the latest peak found (and there is a possibility that the current peak is part of a bigger peak), and this new peak is bigger than the latest peak found so far, then the old peak is replaced by the new, bigger peak. In this way, we are ensured that we pick the largest values of the plot.

Secondly, suppose that we examine the character 'o'. This character has a rather flat $|\Delta\phi|$ - function as depicted in fig.22.

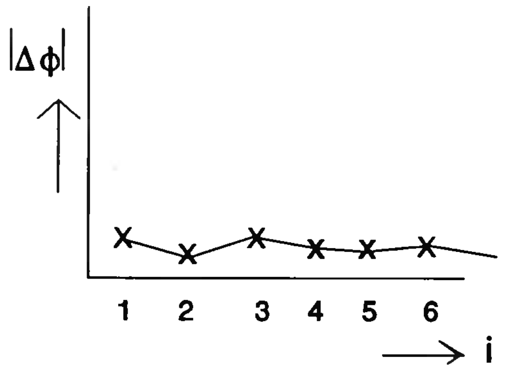


Figure 22 flat plot with no real peaks.

In this case, we can consider the largest values found as false peaks, since there are no peaks at all. We can decide that the peaks are false in this case, when the values are larger than a factor times the mean of all samples. This threshold will save us from detecting false peaks.

5.2.4. Insertion of connecting areas.

In fig. 17 we could make a partitioning of the N $|\Delta\phi|$ -samples, based on the sharp peaks in the plot. We have made the following partition : $\{\phi[1]...\phi[54]\} = \{\{\phi[1]...\phi[20]\}, \{\phi[22]...\phi[35]\}, \{\phi[37]...\phi[44]\}, \{\phi[46]...\phi[50]\}, \{\phi[52]..\phi[54]\}\}$. We have left open spaces of two samples between positions 20-24, 35-37, 44-46 and 50-52. This is done for smoothing purposes. We can explain the reasoning here by taking a look at fig. 23 on the next page.

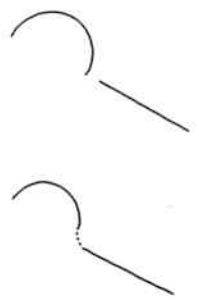


Figure 23 without spacing, difficult matching of two sections (upper), with some space, smooth matching possible (lower).

In the upper case, we have no spacing: the next section starts at the succeeding sample number. Suppose, that the next section is approximated as a straight line, it is possible that the best straight line may lie in a position in which it is hard to match this line with the earlier found best circle (fig.23 upper). This problem can be overcome by taking some space between two succeeding sections. As illustrated in fig. 23 lower, this time it is easy to insert a circle-piece between the two sections. This insertion part must be calculated after all the sections are approximated in the (total) least squares sense. The insertion sections can be easily found, since they are generally arcs of circles. The starting angles of the insertion sections are the ending angles of the previous sections and the final angles are the starting angles of the next section, in order to provide smoothness. The radius can be kept small, which corresponds with a small insertion section. In our procedure, we have chosen for a sample width of three samples. It is important to realize the relations between the various ranges involved. For example, there are N X-Y-samples $[\{X, Y\}_1,\ldots,\{X, Y\}_N]$. This implies that there are N-1 sample of the differential angle $\phi,[\phi_1,\ldots\phi_{N-1}]$.

Consequently, there are N-2 samples of the derivative of this function $[|\Delta\phi|_1,\ldots,|\Delta\phi|_{N-2}]$. It is useful to realize here, that when a peak is found at position i, the insertion section starts at i and ends two samples later. This is because we are interested in a partition of the X-Y-coordinates, and the range of the differential angle functions are different then the range of X-Y-coordinates. To make this all more clear, fig. 24 gives an illustration of the various parameters with their subscripts.

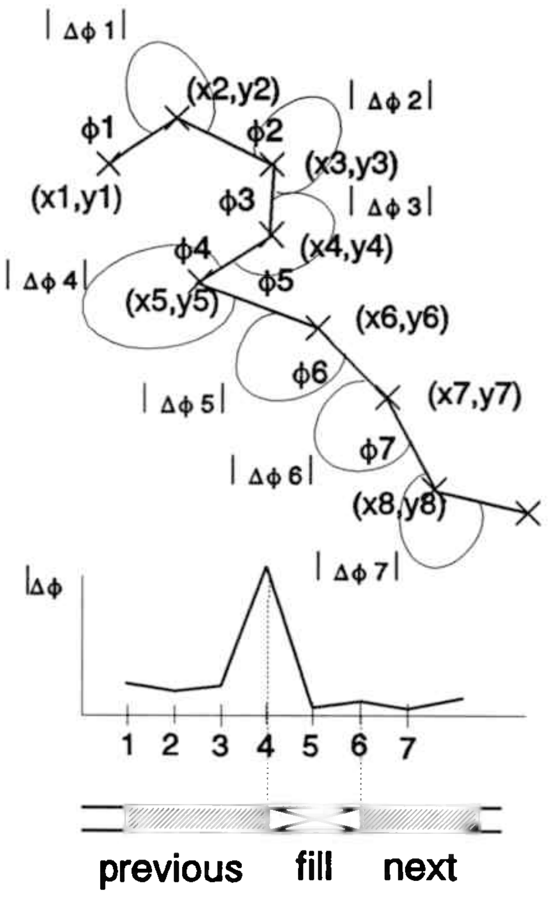


Figure 24 partitioning of a curve into two pieces, with empty space in between.

We can see from figure 24 that a peak occurs at the 4th sample of $|\Delta\phi|$. The ϕ-values involved are ϕ_4 and ϕ_5 and the X-Y-samples involved are (x_4, y_4), (x_5, y_5) and (x_6, y_6). The space between these X-Y-coordinates are members of the insertion section. In the bar at the bottom of figure 24 this fill-space is drawn, referring to the X-Y-coordinates.

Next, the missing links have to be inserted. In most cases this can be easily done by drawing a circle according to figure 25, since starting angle and final angle are known. The radius is half of the distance between the final point of the first section and the starting point of the second section. A parameter which is very dependent on the sort character is the difference between starting angle and final angle of this insertion circle. For some characters this difference may be larger than π, as shown in figure 26.

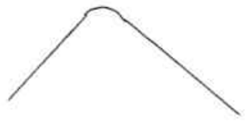


Figure 25 Possible turn for insertion section

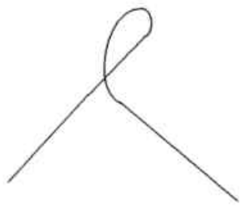


Figure 26 Another possible turn for insertion.

The decision whether to choose between these two options, is made in choosing a maximum difference between starting and final angle of the insertion circle. this parameter is adjusted by hand, which costs extra time. We will see in chapter 6 that the total amount of time to process a character correctly, can in this case increase up to 10 minutes. This problem could be solved by first describing the peaks well, and then approximate the sections in between. This is a different approach which reveals other problems, such as how to describe the peaks, especially, determining starting and final angles of these peak circles.

5.3. An algorithm for processing X-Y-coordinates.

We were so far, able to design a suitable partitioning procedure for the inputed character data. We are also in possession of formulae for calculating the GFDs for each of these sections. (see chapter 2). Now we are in the position to combine these various formulae and procedures to make one data-processing algorithm. First, the character is inputed, after which processing of the data can take place, according to all the mathematics and algorithms discussed.

First, the data are input via the writing tablet. Angles and differential angles are derived from the X-Y-coordinates, because they are required for the partitioning algorithm. Once the partitioning is made, we can start to approximate, for each resulting section, the best circles or straight lines. The decision whether to approximate for best circles or for best straight lines, can be very easy. First, from all the X-Y-coordinates and angles of a particular section, the radius is estimated, with the aid of formula (4-15). When we realize that radius $r = 1/\kappa$, we have a first estimation of the radius. It is clear that if this estimated radius has a large value, we can consider the current section as a straight line and start applying the straight line-estimation procedure. If the radius is smaller than some threshold value, the section is apparently a circle arc. Now the insertion calculations can start, leading to a final split-up of the character. After this partitioning, we know starting and ending angle, starting and ending sample-number of each section. Now we can directly calculate the GFDs from this data, making use of the formulae of chapter 1.A PSD (Program Structure Diagram) of the total data-processing is given in fig. 27. From this point, we can apply this algorithm on each character of the alphabet, starting the character recognition investigations. This part will be treated in the next chapter.

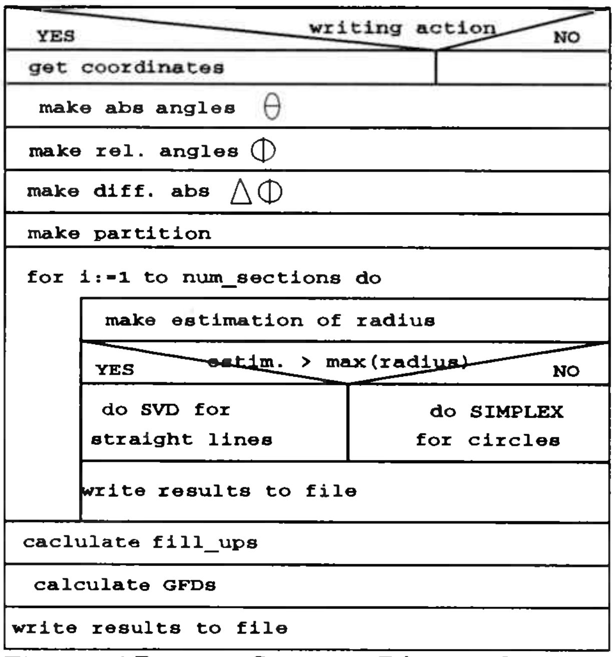


Figure 27 Program Structure Diagram for a GFDs calculating program.

CHAPTER 6. CHARACTER RECOGNITION RESULTS

6.1. INTRODUCTION

In this chapter, we will perform some character recognition experiments. First, the techniques discussed in the previous chapters will be carried out in full, ultimately revealing the GFDs. Next, the obtained GFDs have to be compared. The environment in which all the experiments have been performed is depicted in the figure 28 below. A person's handwriting is written on a writing tablet, which digitizes the written curve and sends absolute and relative angles to the personal computer. All calculations are performed on an AT-286 personal computer .

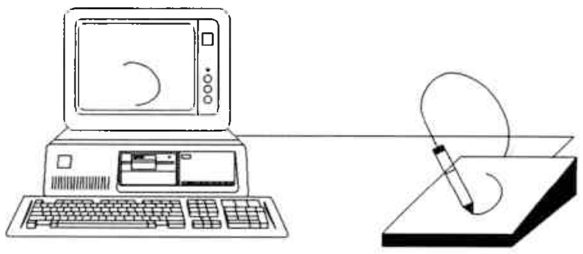


Figure 28 Experimental set-up

Character recognition experiments are often presented on a somewhat ad hoc basis. In order to achieve objective and more clear character recognition, we will develop a frame in which experiments are to be performed. It is stated here that the presented frame is merely introduced to make character recognition more systematic and interpretable. Once this frame is presented, comparison can be made for a set of characters of the Western alphabet. First the whole calculation procedure will be discussed and outcome will be shown.

6.2. CALCULATION PROCEDURE WORKED OUT FOR A SAMPLE CHARACTER

As an example to illustrate the use of the mathematical and numerical techniques, discussed in the previous chapters, we will describe the calculation of the GFDs for a sample character. We will choose the letter 'a'. Of course the procedure applies to all characters of the Western alphabet. The character 'a' was first drawn on the writing tablet, yielding relative angles according to DCC chain codes. Also X-Y-pairs can be obtained from the writing tablet. The X-Y-coordinates of this character can be plotted out as shown in figure 29.

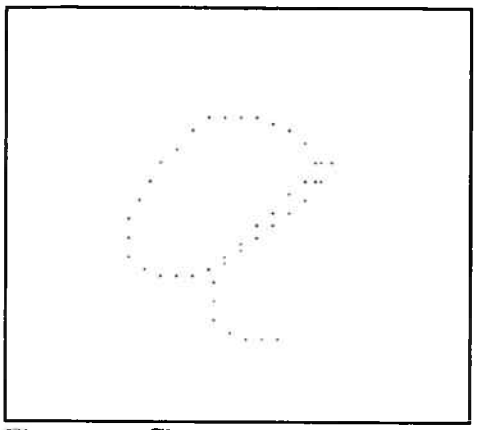


Figure 29 Character as drawn on the writing tablet

The next step will be to make a suitable partitioning of the character. As described before, this partitioning is done using a peak-detection algorithm, which traces peaks in the differential angles. The next figures display the phi-function (the absolute angles along the curve) and its derivative function, both discrete functions.

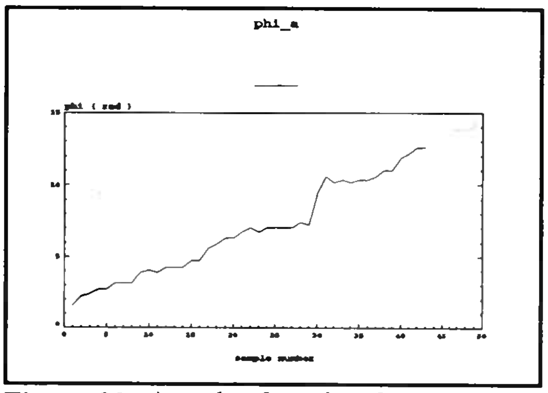


Figure 30 Angular function for the character 'a'

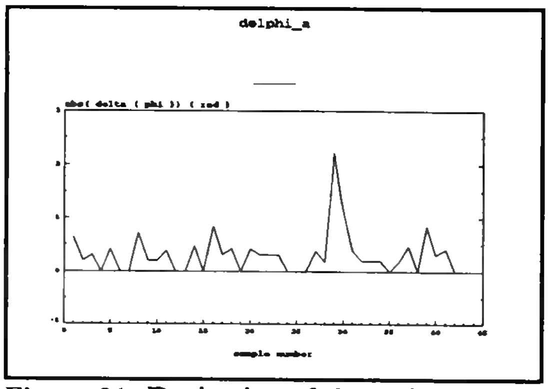


Figure 31 Derivative of the phi-function

Peak-detection will be performed on the difference function (fig. 31). The peak-detection algorithm will first discover 5 peaks at (see fig. 31) sample number 1, 8, 16, 29 and 39. These peaks are sufficiently larger than the mean, otherwise they would be false peaks and be removed). The peak at position 1 will be removed because parts must include at least 3 samples. Next, each part is considered consecutively. For each part the curvature will be approximated as explained earlier. If the curvature is too small (or equivalently : if the estimated radius is too large), then SVD will be used to match a straight line piece to the set of points. If the radius is smaller than a certain value (say 400), then circular matching will be performed. In our example all the parts appear to have relatively small radius, which is why circular matching will be performed on all of them. This agrees with the general feeling that the character 'a' does not consist of straight line-sections. The matching result is shown in the next figure.

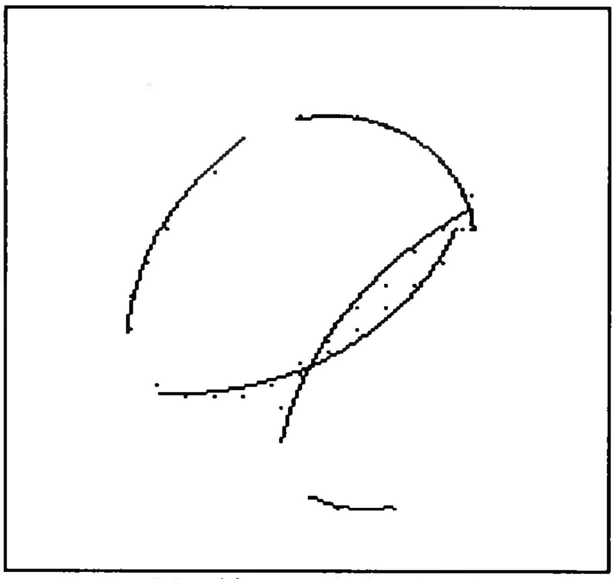


Figure 32 Matching result for the character 'a'.

Note that this character is split up into 5 pieces, because 4 peaks were found. Next, insertion of the missing parts must take place. All insertion parts are small circle parts that have starting angle equal to the preceding part and ending angle equal to the starting angle of the next part. The result after insertion is shown in figure 33 on the next page. It is clear that the proposed method of partitioning the character into sections that consist of circular shapes has no effect on the human ability to recognise the right character. This gives us the feeling that we can proceed with calculations. At this point we can already conclude that the GFDs of characters can be put in a close mathematical form. This implies that the characters can be described in terms of mathematical functions (circles). This is an important feature since we can now proceed to determine GFDs, not in a numerical way, but in an analytical way. We will now start computing the GFDs.

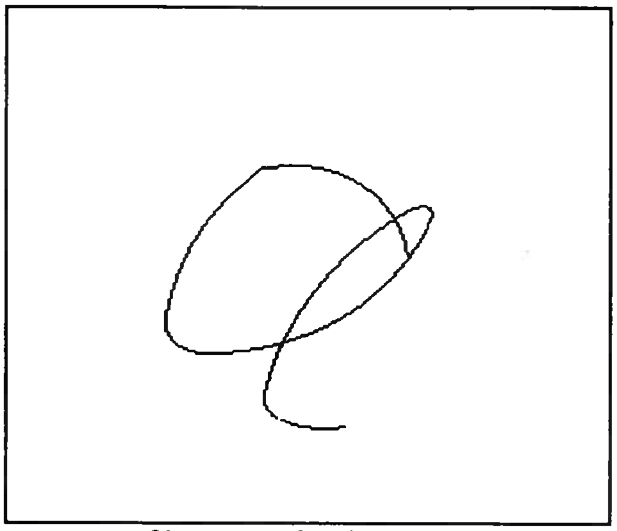


Figure 33 Character after insertion of missing parts.

Since the circle parameters for each section are known, we can calculate the starting and ending distance. Together with the starting end ending angle, we have the straight line in the phi-plane. This plot is drawn in fig. 35. For comparison, the original phi-plot is drawn in figure 34.

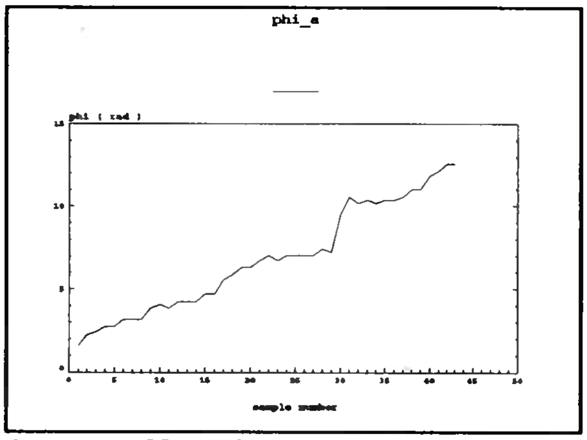


Figure 34 Natural parameters for the original character

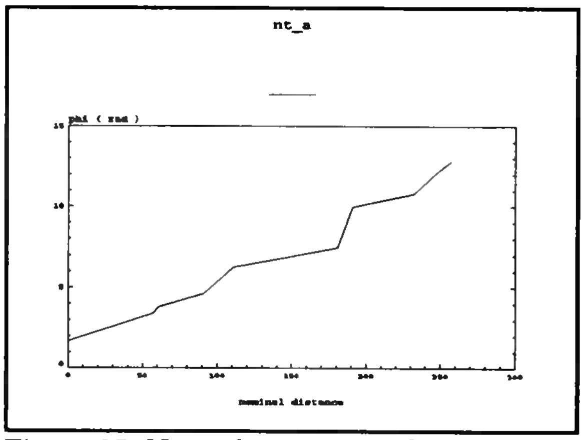


Figure 35 Natural parameters for the matched character.

Next, according to eq. (2-12), a normalised phi-function has to be made, such that the starting angle equals ending angle equals zero. Also the distances are to be normalised, running from 0 to 1. The thus obtained plot is shown in fig.36 on the next page.

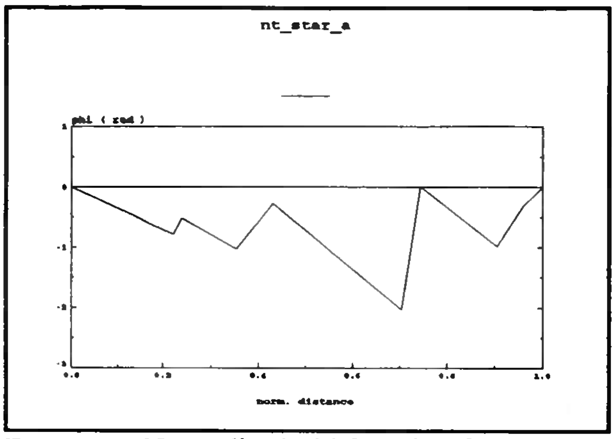


Figure 36 Normalised phi-function for the matched character

The final step is to compute the GFDs for the matched curve, according to eq. (1-20). The resulting spectra are shown in fig. 37 and 38.

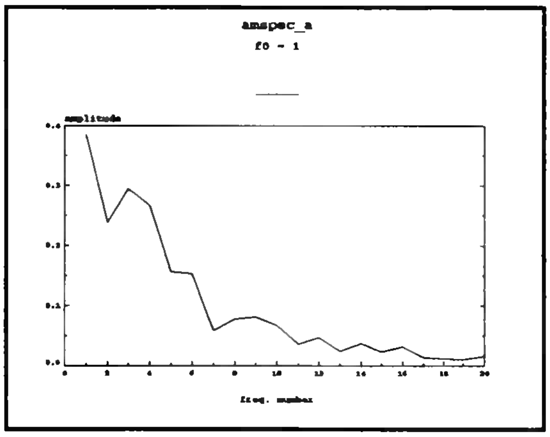


Figure 37 Amplitude spectrum for the matched character

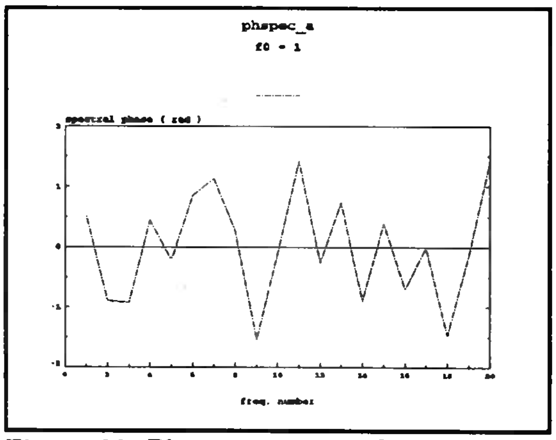


Figure 38 Phase spectrum of the matched character

6.3 Introducing character recognition frame.

At this point we are able to compare the GFDs. But first we have to specify what will be compared. Suppose that a person makes use of the writing tablet and writes down all characters of the alphabet. These characters are stored in computer memory and can later be used for computations. Suppose the calculations are performed for all the characters and he wishes to compare between all the characters. We will call this horizontal character recognition and the process is, in an abstract form, drawn in fig. 39.

abcdefghijklmnopqrstuvwxyz

Figure 39 Horizontal character recognition.

In this way we can tell if a certain method (e.g. spectral comparison with GFDs) distinguishes characters well or not. Suppose now that another person with a different handwriting style writes the characters, recognition might fail, due to big differences in writing styles of human beings. Instead, this person has to store all his characters in memory, after which he can compare characters belonging to his own set. However, for signature verification for example, we are interested in the differences between several persons, when writing the same character. In our character recognition model, we will indicate this by introducing a vertical axis, called the person axis, as can be seen from fig.40.

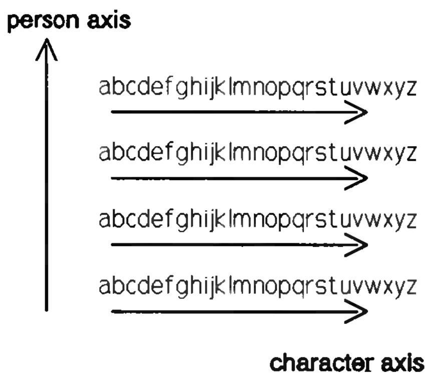

person axis
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
character axis

Figure 40 Introducing second dimension in character recognition

Along the axis we can introduce two measures. First, along the horizontal axis, we introduce the distinguishability, that is a measure for the ability of distinguishing the letters of the alphabet, written by one person. If this distinguishability is high, we can use the method for character recognition purposes, because it each character appears to have its own specific feature. On the other hand, if the distinguishability is low, then the method does not distinguish the characters well, which makes it useless for character recognition. On the vertical axis we introduce the community, which indicates to what extend two or more different persons write certain characters in the same way. If the community is low for some character, then the method is suitable for identification purposes. In this case, certain handwriting styles appears to have their own special features. On the other hand, if the community is high, then it means that each character has some common feature which is independent of the writing style. If both community and distinguishability are high, then we have solved a feature extraction problem. This is an interesting area because it signifies that all characters are substantially different from each other, no matter who writes it. The different areas are drawn in figure 41.

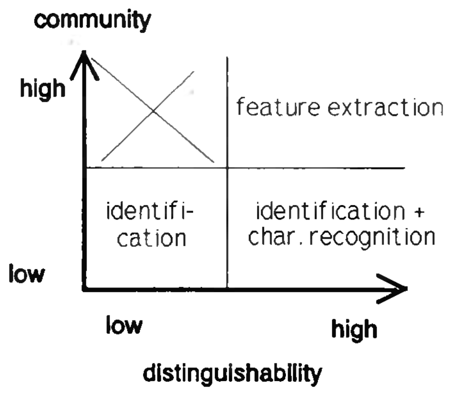


Figure 41 Different research areas placed in the character recognition frame

In this figure, the characters written by one person should be projected along the horizontal axis, while different handwriting styles are accounted for on the vertical axis. Note that the upper left area (where distinguishability is low and community is high) is a region in which the method is not useful : it can neither be used for character recognition, nor for identification.

6.4. GFDs for a set of alphabetic characters.

In this report characters will be compared both in horizontal as in vertical direction. First, several differences will be shown between some characters of the alphabet, written by one person. The whole alphabet is not considered here, because it takes a substantial computing time. Moreover, if some similar looking characters are distinct in GFDs, the use of this method is, at least for those characters, proven. We have chosen to compare the spectral amplitude, because the spectral phase does not contain relevant shape information [1]. The Fourier Coefficients participate in the spectral amplitude, which is why the amplitude is chosen for comparison.

We will first compare the amplitude spectra of the characters 'b', 'd' and 'p'. The smoothed characters together with the amplitude spectra are shown in subsequent figures.

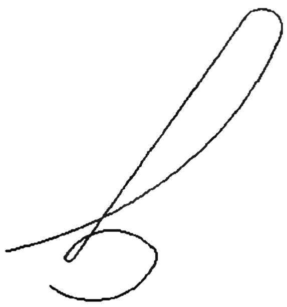


Figure 42 Smoothed character 'b'.

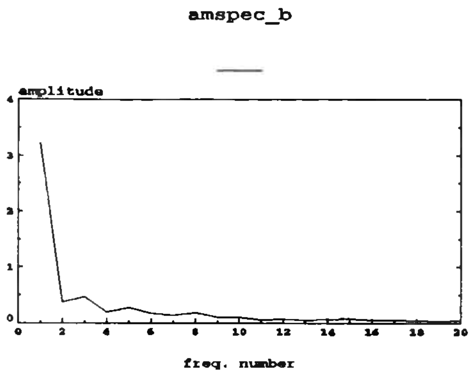


Figure 43 Amplitude spectrum for 'b'.

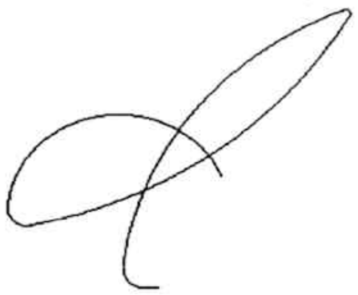


Figure 44 Smoothed character 'd'.

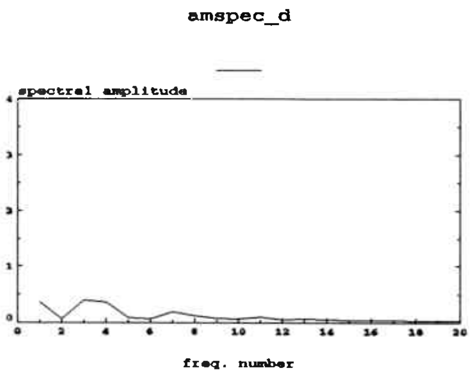
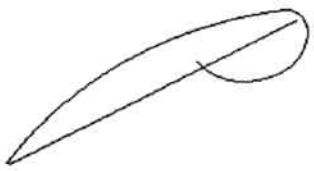


Figure 45 Amplitude spectrum for smoothed 'd'.



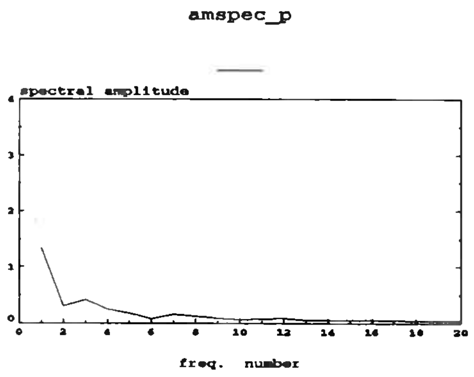


Figure 47 Amplitude spectrum for smoothed 'p'.

Figure 46 Smoothed character 'p'.

From these plots we can see that the characters clearly differ in amplitude. The character 'b' has larger values than the character 'p' at the low frequency-region. The character 'd' appears to be completely different. So we can see that the amplitude shows good distinguishability for these characters.

Next, we will examine the influence of different handwriting styles. For example, we will consider the characters 'h' and 'k'. These characters were written by three persons. Each row of figures will refer to one test person. Only the smoothed character and the amplitude spectrum for each of the three persons is considered here. First, the letter 'h' is examined.

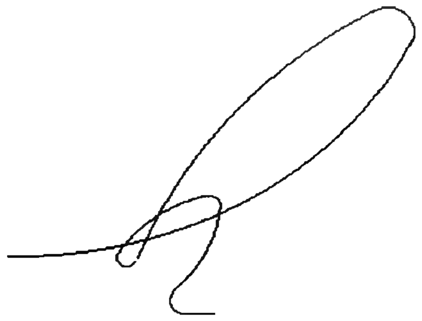


Figure 48 Character written by person 1.

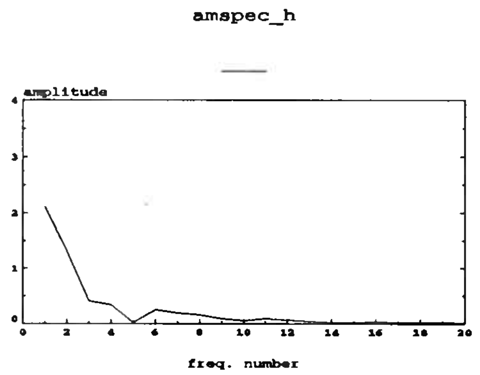


Figure 49 Amplitude spectrum for letter h' by person 1.

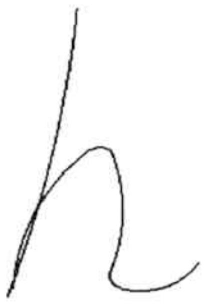


Figure 50 Same character written by person 2.

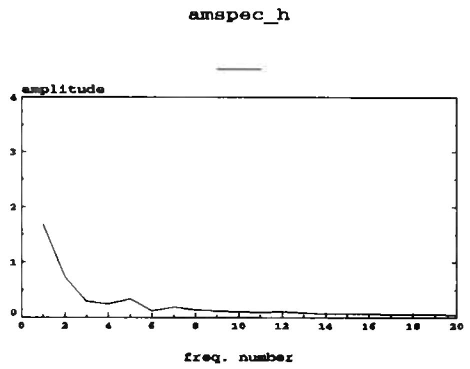


Figure 51 Amplitude spectrum for letter 'h' by person 2.

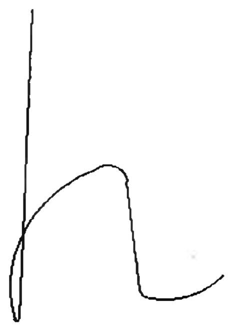


Figure 52 Same character written by person 3.

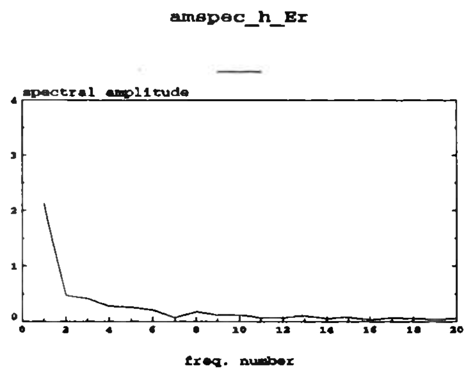


Figure 53 Amplitude spectrum for person 3.

We can see that although the characters are written in a clearly different way, the amplitude spectra are quite similar. This implies in the previously given framework, that the community (at least for this character) is high. To illustrate this with one more example we consider the character 'k'.

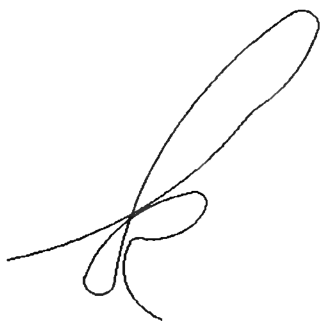


Figure 54 Character 'k' written by person 1.

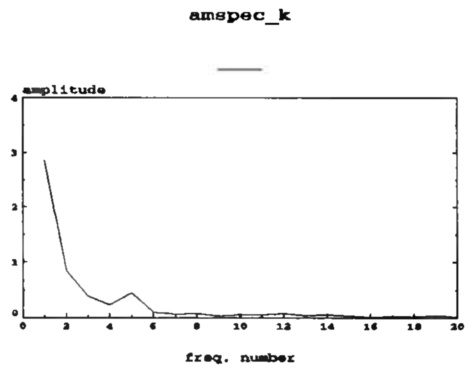


Figure 55 Amplitude spectrum for person 1.

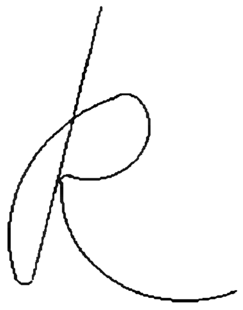


Figure 56 Same character written by person 2.

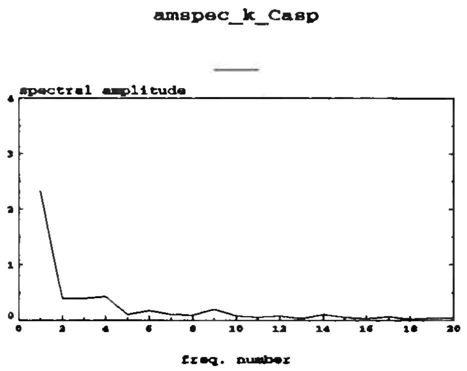


Figure 57 Amplitude spectrum for person 2.

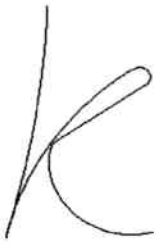


Figure 58 Same character written by person 3.

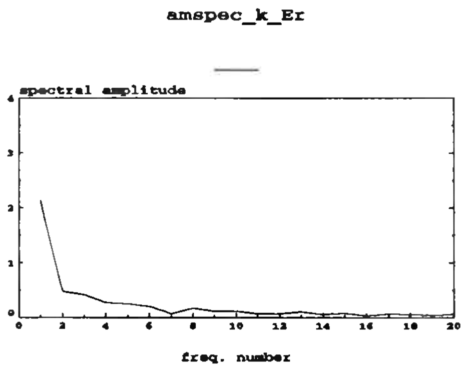


Figure 59 Amplitude for person 3.

Again we can see that the amplitudes are quite common. We have at this point shown that characters that have similar shapes ('d', 'p' and 'k'), possess more different amplitude spectra. This implies that GFDs are suitable for character recognition: they have shown to distinguish characters that have similar shapes. At the same time it appeared that although some characters are written differently by different persons, the amplitude spectrum shows reasonable similarity. This indicates that GFDs are not suitable for identification.

6.5. COMPUTATION TIMES.

At this point we have to spend a few words on the calculation times in the experiments. The fastest numerical procedure is the partitioning algorithm: it only takes a few seconds. The next fastest procedure is the singular value decomposition, which also takes a few (1 to 2) seconds, due to the cubic convergence of the method (see [11]). The SIMPLEX numerical method for solving the non-linear total least squares problem costs a larger amount of computing time. Depending on the number of coordinate pairs, the accuracy of the initial estimates (eq.4-14 and 4-15) and the terminating accuracy ϵ (eq. 4-16), it can take at about 30 seconds to obtain the desired best circle for one section. In all experiments the numerical accuracy was set to ϵ = 1E-6. If there are 6 segments which are (in the worst case) all circles, then the curve-fitting time will be 6 times 30 seconds = 3 minutes. Next, insertion of the missing links takes place, as described in section 5.2.4, which takes at about ten seconds. Then the GFDs have to be computed. Since we have determined the formulae (eq. 2-7 and 2-21), these are also computed fast. Finally, we want to make plots of the functions ϕ^*, $|\Delta\phi^*|$, A_k, and α_k. These graphs are made in 'Charting Gallery', a plotting utility which is being installed on the personal computers at the laboratory. This plotting costs minutes, because the data has first to be read from data files, the plot has to be made graphically and especially print outs cost minutes of time per plot. We can state that the total computing time, from a test person's writing until the obtained plots of the GFDs can cost at least 5 to 6 minutes.

It has to be mentioned here, that these procedures take as much ime as indicated above, unless no parameters have to be adjused. As mentioned in section 5.2.4., the insertion circle may have a more complex shape than the single curve (see fig. 25 and fig. 26). As mentioned, the right choice is made by adjusting the maximum allowable difference in starting and final angle of the insertion circle. In a majority of the cases the program this maximum different angle is adjusted well. In some cases this maximum angle difference between starting and final angle of the insertion circle, has to be adjused by hand, after which the program has to be executed again. This costs extra time and also the GFDs have to be computed again. The total processing time can in these cases increase up to 10 minutes for a character. This problem could be solved by first describing the peaks well, and then approximate the sections in between. As mentioned earlier, this is a different approach which reveals other problems, such as how to describe the peaks. This approach has not been considered in this work.

CHAPTER 7. CONCLUSIONS AND RECOMMENDATIONS.

7.1. CONCLUSIONS.

We can conclude that a new method for character description by Generalised Fourier descriptors has been suggested. This method 'polishes' the character first, before GFDs are computed. Polishing here means that the characters are first fitted into the best straight lines and circles, in the Total Least Squares sense. After this curve-fitting, GFDs are calculated, based on the expressions that are known for straight line sections and arcs of circles. This method has been applied on several characters to compare the corresponding amplitude spectra. Also handwritings of different persons have been involved, to show the differences between several handwriting styles. From these experiments it has been shown that the characters used in the experiments have good distinguishing qualities in horizontal direction, that is, GFDs for different characters are also clearly different. In vertical direction (that is with several test persons) the results found in the previous chapter show great similarity in spectral amplitude for obviously different handwritings. Therefore, GFDs can be used to distinguish several characters, which may even have similar shapes. GFDs are not suitable for identification at this point, because the several different handwritings appear to have similar spectral amplitudes.

7.2. RECOMMENDATIONS.

Of course, these conclusions are not checked for every single character of the alphabet. We have chosen characters that have similar shapes to illustrate that these characters have quite different spectral amplitudes. Also the number of test persons was restricted to 3, but from the experiments it can be seen that the handwritings were quite different. Again the method has been proven to be useful in this case, bacause the spectral amplitudes were fairly similar.

To investigate this method thoroughly, we have to include a dozen of test persons, where each test person writes all characters of he alphabet. This would mean that, say, 10 times 26 = 260 characters have to be written. From section 6.5 we know that it takes at about 10 minutes to process one character correctly, that is, computer processing time plus proper adjustments of parameters and making plots. This implies that we would need more than 40 hours of processing time, just for inputing the characters. Then we still have to compare the output.

It is recommended here that this thorough study should be carried out, or at least some more experiments should be added, to qualify the use of this method. Also comparison should be made with the 'sampled line' method, to see how the GFDs have changed after curve-fitting. Finally, if much time is available, also other GFDs-parameters, such as a_k and b_k should be considered.

Finally, an other approach of the partitioning problem should be investigated. As mentioned earlier, we can start the partitioning procedure by first describing the peaks well, after which the other segments are to be inserted. In this description we do not have the problem of adjusting the maximum allowable difference in starting and final angle at the peaks by hand. On the other hand, the key problem here is how to describe the peaks. For example the starting and final angle of the peak section must be specified.

REFERENCES

[1] ZAHN, C.T., and ROSKIES, R.Z., 'Fourier Descriptors for plane closed curves ', IEEE trans. Computers, Vol. C-21, pp.269-281, 1972.

[2] FREEMAN, H., ' On the encoding of arbitrary geometric figures', IRE Trans. Electron. Comput., Vol. EC-10, pp.260-268.

[3] ARNBAK, J.C., BONS, J.H. and VIEVEEN, J.W., ' Graphical correspondence in Electronic-Mail networks using Personal Computers', IEEE Sel. Areas Commun., 1989, SAC-7, pp.257-267.

[4] VIEVEEN, J.W. and PRASAD, R., ' Generalised Fourier Descriptors for use with line-drawings and other open curves', Proc. 6th Scandinavian Conference on Image Analysis, Oulu, Finland, June 1989, pp. 820-827.

[5] PRASAD, R. VIEVEEN, J.W., BONS, J.H. and ARNBAK, J.C., ' Relative vector probabilities in differential chain coded line-drawings', Proc. IEEE Rim Conference on Communication, Computers and Signal Processing, Victoria, Canada, June 1989, pp.138-142.

[6] LIU, K. and PRASAD, R,' On the quantisation distortion and coding efficiency in line-drawing graphics transmission using differential chain coding, ' Proc. IEEE International Conference on Communications (ICC'90), Atlanta, USA, April 1990, pp. 325A.2.1.-325A.2.5.

[7] WEYLAND, N.B.J. and PRASAD, R.,' Characterisation of line drawings using generalised Fourier Descriptors',Electronics Letters, October 1990, Vol. 26, No. 21, pp.1794-1795.

[8] BERNARD, F.,' Datacompressie van facsimiledocumenten t.b.v. transmissie via Electronic-Mail netwerken',graduation report TU-Delft, January 1991.

[9] STRANG, G., ' Linear Algebra and its applications', Academic Press, 2nd edition 1980, Ney York.

[10] RAO, S.S., ' Optimization, theory and applications', Wiley Eastern, 1978, New Delhi.

[11] GOLUB, G. and VAN LOAN,C.F.' An analysis of the Total Least Squares problem ', Siam Journ. Numer. Anal., Vol.17, No.6, pp. 883-893, December 1980, pp.883-893.

[12] GOLUB, G.H. and REINSCH, C., ' Singular Value Decomposition and Least Squares solutions ',Numer. Math. 14, pp.403-420, 1970.

[13] GOLUB, G. and KAHAN, W., ' Calculating the Singular Values and Pseudo-inverse of a matrix', J. Siam Numer. Analys. Ser.B, Vol. 2, No.2, pp. 205-223, 1965, pp.205-223.

[14] GOLUB, G. and VAN LOAN, C.F., ' Matrix Computations ',John Hopkins, Baltimore, 1983.

[15] TAPPERT, C.C., SUEN, C.Y. and WAKAHARA, T., The state of the Art in On-line Handwriting Recognition', IEEE Trans. on Pattern Anlysis and Machine Intelligence, Vol. 12, No.8, August 1990, pp.787-808.

[16] WEYLAND, N.B.J. and PRASAD, R., 'Criterion for characterisation of line-drawings using Generalised Fourier Descriptors', accepted for presentation at the 7th Scandinavian Conference on Image Analysis, 13-16 August 1991, Aalborg University. Denmark.

APPENDIX A : Computation of the singular value decomposition

Let A be a real mxn matrix with m≥n where m denotes the number of rows and n denotes the number of columns. A can be factorized as :

$$A = U \Sigma V^T \tag{A1}$$

where

$$U^T U = V^T V = I_n \quad and \quad \Sigma = diag(\sigma_1, \dots, \sigma_n) \tag{A2}$$

The matrix U consists of n orthonormalized eigenvectors associated with the n largest eigenvalues of AA^T, and the matrix V consists of the orthonormalized eigenvectors of A^TA. The diagonal elements of Σ are the non-negative square roots of the eigenvalues of A^TA and are called the singular values of A. We shall assume that

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0 \tag{A3}$$

If rank(A) = r then $\sigma_{r+1} = \sigma_{r+2} = \dots = \sigma_n = 0$.

We want to point out here that it is easier to calculate the eigenvalues of A^TA since they are the squares of the singular values of A. However, the computationn of A^TA leads to less numerical accuracy. For example, let

$$A = \begin{bmatrix} 1 & 1 \\ \beta & 0 \\ 0 & \beta \end{bmatrix} \quad , \quad A^T A = \begin{bmatrix} 1+\beta^2 & 1 \\ 1 & 1+\beta^2 \end{bmatrix} \tag{A4}$$

$$\sigma_1(A) = \sqrt{(2+\beta^2)} \quad , \quad \sigma_2(A) = |\beta| \tag{A5}$$

If $\beta^2 < \epsilon_0$, the machine precision, we would get instead

$$A^T A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad and \quad \overline{\sigma_1} = \sqrt{2} \quad , \quad \overline{\sigma_2} = 0 \tag{A6}$$

Next, we will briefly discuss the steps to be taken to obtain the decompostion.

The first step is to bring the matrix A into a bidiagonal form . This is accomplished by using

Householder transformation. We introduce the two Householder matrices as P and Q, which have the following form :

$$P^{(k)} = I - 2\mathbf{x}^{(k)}\mathbf{x}^{(k)^{\mathrm{T}}} \qquad (k = 1,2,\ldots n)$$
$$Q^{(k)} = I - 2\mathbf{y}^{(k)}\mathbf{y}^{(k)^{\mathrm{T}}} \qquad (k = 1,2,\ldots n\text{-}2) \tag{A7}$$

Householder matrices are essentially projection matrices, which can therefore be used to create zeroes in a vector **x** or **y**. Premultiplying a matrix A n times (each for one column) with the Householder matrix P, creates zeroes in the column under consideration. Postmultiplying the matrix A with the Householder matrix Q n times creates zeroes in the rows, likewise. This will eventually lead to a bidiagonal matrix of the following form :

$$P^{(n)}\ldots P^{(1)}AQ^{(1)}\ldots Q^{(n-2)} = \begin{bmatrix} q_1 & e_2 & 0 & . & . & 0 \\ 0 & q_2 & e_3 & 0 & . & 0 \\ . & & . & . & . & . \\ . & & & . & . & 0 \\ . & & & & . & e_n \\ . & & & & & q_n \\ 0 & 0 & 0 & . & . & 0 \\ . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & 0 \end{bmatrix} \equiv J^{(0)} \tag{A8}$$

An essential property of the Householder projection is that the singular values remain the same. This implies that the matrix $J^{(0)}$ has the same singular values as A. We can therefore compute the singular values of the bidiagonal matrix $J^{(0)}$ instead. Calculation of these singular values can noew proceed with the aid of some QR-like algorithm. This is a numerical process, which iteratively diagonalizes the bidiagonal matrix $J^{(0)}$:

$$J^{(0)} \rightarrow J^{(1)} \rightarrow \;.\;.\;.\; \rightarrow \Sigma$$

and where

$$J^{(i+1)} = S^{i} J^{i} T^{i} \tag{A10}$$

and $S^{(i)}$ and $T^{(i)}$ are orthogonal matrices. The matrices $S^{(i)}$ and $T^{(i)}$ are chosen such that the matrix :

$$M^{(i)} = J^{(i)^T} J^{(i)} \tag{A11}$$

converges to a a diagonal matrix, and all $J^{(i)}$ are bidiagonal. The transition from $j^{(i)}$ to $J^{(i+1)}$ is achieved, using Given's Rotations. A givens Rotation matrix has the form:

$$S_k = \begin{bmatrix} 1 & 0 & & & & & & & \\ 0 & . & & & & & & & \\ & & . & & & & & & \\ & & & 1 & & & & & \\ & & & & \cos\theta_k & -\sin\theta_k & & & \\ & & & & \sin\theta_k & \cos\theta_k & & & \\ & & & & & & 1 & & \\ & & & & & & & . & \\ & & & & & & & . & 0 \\ & & & & & & & 0 & 1 \end{bmatrix} \tag{A12}$$

To illustrate the effect of Givens Rotation, we will examine the following 2x2-case where we see that the lower left element in the A-matrix becomes zero. :

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} a_{q,1} & a_{q,2} & \cdot & a_{q,N} \\ a_{q+1,1} & a_{q+1,2} & \cdot & q_{q+1,N} \end{bmatrix} = \begin{bmatrix} a'_{q,1} & a'_{q,2} & \cdot & a'_{q,N} \\ 0 & a'_{q+1,2} & \cdot & a'_{q+1,N} \end{bmatrix} \quad \text{A13)} \quad ($$

if

$$\theta = -\arctan\left[\frac{a_{q+1,1}}{a_{q,1}}\right] \tag{A14}$$

If we now want to delete the right top element of the matrix, we will apply Givens Rotation again, with proper choices for θ. However, then the lower left element will become non-zero again. Therefore we apply the Givens rotation again to make it zero again. Then the upper right element will become nonzero. Again Givens Rotation will be applied etc. This process is called "chasing". It can be shown that eventually, the non-diagonal elements will become smaller ans smaller. This process has in most cases a cubic convergence, which means that after every chase the non-diagonal elements will become a power -3 smaller. This process can be carried out until the non-diagonal elements have reached some value ϵ.

For a more detailed coverage of the singular value decomposition and its computation, we refer to [11]-[14]. Golub and Reinsch [12] have presented an Algol program is given which computes the singular value decomposition. This source code is converted to a corresponding Pascal code and it is printed out on the next pages. The Pascal code will be applied for curve fitting on handwritten curves in chapter 3 and further.

APPENDIX B : LISTINGS OF PASCAL PROGRAMS

1 : **The_end.pas.**

This program was first originally written by Jan Vieveen. It was used by Mr. Vieveen to do character recognition experiments. The program was modified in order to obtain X-Y-coordinate pairs from the writing tablet. These X-Y-coordinate-pairs will be used by the following Pascal-programs Match2.pas and Calcspec.pas.

```
{$B-}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$I-}    {I/O checking on}

program SIGN_ON;
{$R+,V-,F-,B-}
{ M 64000,1,655360 }

{
*************************************************
* Turbo TOTAAL, T.U.Delft                       *
*  Copyright       1988    J.W.Vieveen          *
* Program  "The end"                            *
*                                               *
*                              vivi             *
*************************************************
}


Uses
  Dos,
  Printer,
  Graph,  BOX004,  tgg, signbox,  Crt;

const
maxdcc      = 40;
maxdccshr  = 20;
maxint      = 10000;

type
mask        = array[0..maxdcc] of integer;
fmask       = array[0..maxdccshr,1..2] of real;

masks = record
maskarray   : array[0..127] of mask;
maxarray    : array[0..127] of integer;
fouriermask : array[0..127] of fmask;
end;

charrec = record
dcc      : masks;
waittime : integer;  { time to wait in milliseconds }
end;

var
memchars                    : charrec;
memchar                     : mask;
memfour                     : fmask;
mcfile                      : file of charrec;
memtime, person             : integer;
resstr,fresstr,dummystr     : anystr;
dum                         : real;
to_file                     : boolean;

tf, xy, ang, fd             : text;
chararr                     : array[1..3] of char;
totalen                     : array[1..10] of real;

farray                      : array[1..3,1..7,0..maxdcc] of real;


label overnieuw;


procedure fourier(memchar : mask;var  memfour: fmask);
var i,j : integer;
sinus,cosinus  : real;

begin
fillchar(farray,sizeof(farray),0);
fillchar(totalen,sizeof(totalen),0);

for i:=0 to maxdcc do
   begin
   totalen[1]:=totalen[1]+memchar[i]/maxint*pi;
   end;

if totalen[1]=0 then totalen[1]:=1;

for i:=0 to maxdcc do
   begin
   totalen[5]:=totalen[5]+memchar[i];
   farray[1,1,i]:=totalen[5];
   end;

for i:=0 to maxdcc do
   begin
   farray[1,1,i]:=(farray[1,1,i]/totalen[5] - i/maxdcc)*2*pi;
   end;

fillchar(totalen,sizeof(totalen),0);
totalen[1]:=0;
assign(fd,'a:\chrdat\fd.dat');
rewrite(fd);

for i:=0 to maxdcc do
   begin
   for j:=0 to maxdcc do
       begin
       farray[1,2,i]:=farray[1,2,i]+farray[1,1,j]*sin(2*pi*i*j/succ(maxdcc));
       farray[1,3,i]:=farray[1,3,i]+farray[1,1,j]*cos(2*pi*i*j/succ(maxdcc));
       end;
{
   if (i>0) then
       begin
       farray[1,2,i]:=farray[1,2,i]/pi/i;
       farray[1,3,i]:=farray[1,3,i]/pi/i;
       end;
}
   farray[1,4,i]:= farray[1,2,i];
   farray[1,5,i]:= farray[1,3,i];
   end;

for i:=0 to maxdccshr do
   begin
   farray[1,6,i]:=sqrt(sqr(farray[1,4,i])+sqr(farray[1,5,i]));  { AMPL }

   {write(lst,#15,farray[1,6,i]:6:2);}

   if farray[1,4,i]=0 then
       begin
       if farray[1,5,i]>0 then farray[1,7,i]:=pi/2
       else farray[1,7,i]:=3*pi/2;
       end
   else farray[1,7,i]:=arctan(farray[1,5,i]/farray[1,4,i]);     { PHASE }
   if farray[1,4,i]<0 then farray[1,7,i]:=farray[1,7,i]+pi;
   if farray[1,7,i]<0 then farray[1,7,i]:=farray[1,7,i]+2*pi;
   memfour[i,1]:=farray[1,6,i];  memfour[i,2]:=farray[1,7,i];
   writeln(fd,memfour[i,1]:6:2,'    ',memfour[1,2]:6:2);
   end;
   close(fd);

end;


function calctime: integer;
var
  recpack:           registers;           {assign record}
begin
recpack.ax := $2c00;
intr($21,Dos.Registers(recpack));                      {call interrupt}
calctime:= (recpack.dx shr 8)*100 + (recpack.dx and $ff);
end;

procedure resettime;
begin
memtime:=calctime;
end;

function notime: boolean;
```

```
var g : anystr;
begin
notime := memchars.waittime<((calctime-memtime+6000) mod 6000);
end;

function readdatabuffer : byte;
begin
if writepointer=readpointer then
   repeat until (writepointer<>readpointer)or notime;
if readpointer<>writepointer then
   begin
   resettime;
   readdatabuffer:=databuffer[readpointer];
   readpointer:=succ(readpointer);
   if (readpointer>maxdatabuffer) then readpointer:=1;
   if stopinterrupt then
       begin
       if (readpointer>writepointer) then
          full:=(readpointer-writepointer<switchdatapointer){<....WR>}
       else                                                 {<R....W>}
          full:=(readpointer+maxdatabuffer-writepointer<switchdatapointer);
       if not full then
           begin                                         {CTS low}
           port[RS232_1offset+modctrreg]:=$0A;
           stopinterrupt:=false;
          end;
       end;
   end;
end;


procedure isr; {interrupt service routine}
interrupt;
begin
inline($FB);       { cli              }
databuffer[writepointer]:=port[RS232_1offset+recdatreg];     { store data in buffer }
writepointer:=succ(writepointer);
if writepointer>maxdatabuffer then writepointer:=1;
if (writepointer mod samplepointer = 0) then             {save time, sample!}
   begin
   if (readpointer>writepointer) then
       full:=(readpointer-writepointer<switchdatapointer)  {<....WR>}
   else                                                  {<R....W>}
      full:=(readpointer+maxdatabuffer-writepointer<switchdatapointer);
   if full then
       begin                                             {CTS low}
       port[RS232_1offset+modctrreg]:=$08;
       stopinterrupt:=true;
       end;
   end;

port[piccmd]:=$20;                                       {end of interrupt}
end;

procedure getvector; { get old vector number with DOS-interrupt }
begin
getintvec($0C,vect);
end;

procedure setvector_new; { set new vector number for user interrupt }
begin                    { with offset of isr }
setintvec($0C,@isr);
end;

procedure setvector_old; { set old vector number for old interrupt}
begin
setintvec($0C,vect);
end;



procedure setreg;
var       portinit       :byte;

begin
```

```
  port[picmask]:=(port[picmask] and $EF); { enable async-com interrupt /F7/}
  inline($FA);                            { cli }
  getvector;                              { get old vector number }
  setvector_new;                          { set new vector number }
  inline($FB);                            { sti                   }
  port[RS232_1offset+modctrreg]:=$0A;     { DTR,OUT2 signals active   }
  port[RS232_1offset+intenreg]:=$01;      { enable rec data available int }
end;

procedure restorevar;                     { restore old values }
begin
setvector_old;
port[picmask]:=(port[picmask] or $10);
port[RS232_1offset+modctrreg]:=$08;       {CTS low}
end;


procedure readfromtablet;
begin
resettime;
ipbcounter:=1;
repeat
   ind1:=ipbcounter div recsize +1;
   ind2:=ipbcounter mod recsize +1;
   ipbcounter:=succ(ipbcounter);
   inputbufferpointer^[ind1,ind2]:=readdatabuffer;
until notime or (ipbcounter=recsize*norec) or keypressed;
end;


procedure initialisation;
begin
if initcom(1,9600,0,1,8) then begin end;
writepointer:=1;
readpointer:=1;
fillchar(databuffer,sizeof(databuffer),0);
stopinterrupt:=false;
setreg;
filename:='';
for i:=0 to 360 do
   hoeken[i]:=0;
checkbreak:=true;
initialize;   { graphics }
restorecrtmode;
filesloaded:=0;
start; welkom;
ipbcounter:=0; str1counter:=0; str2counter:=0; str3counter:=0;
str4counter:=0;  str5counter:=0;
fillchar(filenames1,sizeof(filenames1),0);
fillchar(filename,sizeof(filename),0);
keuzepos:=1;
fillchar(memchars,sizeof(memchars),0);
fillchar(memfour,sizeof(memfour),0);
to_file:=FALSE;
end;

function free_mem : real;
begin
if maxavail<0 then free_mem:=maxavail+65536.0
else free_mem:=maxavail;
end;

procedure checkfree_mem(bytes_necc : longint; outputtext : str80);
var outputstr : str80;
begin
if bytes_necc>free_mem then
   begin
   outputstr:='HALT: Computer has not enough memory!. Buffer: '+outputtext;
   commandline(outputstr);
   beep; halt;
   end;
end;

procedure allocate_buffers;
begin
```

```
checkfree_mem(recsize*norec,'Inputbuffer');
new(inputbufferpointer);
checkfree_mem(recsize*norec,'Str1.buffer');
new(str1bufferpointer);
checkfree_mem(recsize*norec,'Str2.buffer');
new(str2bufferpointer);
end;

procedure init_buffers;
begin
commandline('Initialisation buffers...');
for j:=1 to norec do
   begin
   if (j) mod 10=0 then
       begin
       gotoxy(30,wherey); write(j div 10);
       end;
   for i:=1 to recsize do
       begin
       inputbufferpointer^[j,i]:=0;
{      str1bufferpointer^[j,i]:=0;
       str2bufferpointer^[j,i]:=0;
       str3bufferpointer^[j,i]:=0;
       str4bufferpointer^[j,i]:=0;
       str5bufferpointer^[j,i]:=0;}
       end;
   end;
end;



function uppercase(input : byte) : boolean;
begin
uppercase := (char(input) in ['A'..'Z']);
end;

function printachar(input : byte) : char;
begin
if input>31 then printachar:=char(input) else
printachar:=' ';
end;


function makenatpar(ibufferpointer : bufferpointer; counter: integer; var vulchar : mask)
var
factor,radius,npx,npy,angle,memangle: real;
i,j,filldcc,dx,dy,oldvec,xpos,ypos,
maximum                              : integer;
by                                   : byte;
const    maxi                        = 2;

begin
assign(xy,'a:\chrdat\xy.dat');
assign(ang,'a:\chrdat\ang.dat');
rewrite(ang);
rewrite(xy);

factor:=(str1counter-6)/maxdcc;
radius:=10.5*factor;
maximum:=0;

for i:=1 to maxi do
   begin
   j:=0; filldcc:=-1; oldvec:=0;  angle:=0; memangle:=0;
   xpos:=0;  ypos:=0;  npx:=0;  npy:=0;
   repeat
       by :=ibufferpointer^[j div recsize + 1,j mod recsize + 1];
       case by of
           pndnch: begin
                   inc(j,4);
                   end;
           pnupch: ;
           else begin
{           writeln(lst,by:10,char(by),oldvec:10,dx:10,dy:10);}
               tablevector(by,oldvec,dx,dy);
```

```
                inc(xpos,dx);  inc(ypos,dy);
{               write(lst,'    x,y,j,= ',xpos:3,ypos:3,j:3);}
                if i=maxi then writeln(xy,xpos,'         ',ypos);
                if (sqr(xpos-npx)+sqr(ypos-npy) >= sqr(radius)) then
                    begin
                    if (xpos=npx) then
                        begin
                        if (ypos>npy) then angle:=pi/2
                        else angle:=-pi/2;
                        end
                    else
                        begin
                        angle:=arctan( (ypos-npy)/(xpos-npx) );
                        if (xpos<npx) then angle:=angle+pi;
                        npx:=npx+cos( angle )*radius;
                        npy:=npy+sin( angle )*radius;
                        end;
                    if (angle-memangle)> pi then
                        repeat
                            angle :=angle-2*pi;
                        until (angle-memangle<=pi) and (angle-memangle>=-pi);
                    if (angle-memangle)<-pi then
                        repeat
                            angle :=angle+2*pi;
                        until (angle-memangle<=pi) and (angle-memangle>=-pi);
                    if filldcc>=0 then
                        vulchar[filldcc]:=round((angle-memangle)/pi*maxint);
                    if (filldcc>=0) and (i=maxi) and (abs(vulchar[filldcc])>maximum) then
                        maximum:=abs(vulchar[filldcc]);
                    if i=maxi then  writeln(ang,angle);
                    memangle:=angle;
                    inc(filldcc,1);
                    end;
                end;
            end;
        inc(j,1);
    until (j>=counter) or (filldcc>maxdcc);
    if (j<counter) then radius:=radius*counter/j;
    if (filldcc<maxdcc) then radius:=radius*filldcc/maxdcc;
    end;
    close(xy);
    close(ang);
end;

procedure showmaxtime;
var   g : anystr;
begin
setfillstyle(1,0);
bar(round(maxx*3/4+1),round(maxy*3/4+20),maxx-1,round(maxy*3/4+30));
setcolor(7);
str(memchars.waittime*10,g);
g:=' Msec. to wait:'+g;
outtextxy(round(3/4*maxx), round(3/4*maxy)+20,g);
end;

procedure changetime;
var  ch: char;
begin
setfillstyle(1,15);
bar(1,maxy-25,maxx-1,maxy-1);
outtextxy(22, maxy-13,
   'CHANGE TIME: + = plus 10 msec,  - = minus 10 msec, Q = quit time adjustment');
setfillstyle(1,0);
repeat
   ch:=upcase(readkey);
   case ch of
       '+':inc(memchars.waittime);
       '-':if (memchars.waittime>1) then dec(memchars.waittime) else beep;
       'Q': ;
       else beep;
       end;
   showmaxtime;
until (ch='Q');
end;
```

```
procedure showchar( ipchar : mask; color : integer; wis : boolean);
var  i : integer;
factor : real;
begin
factor:=((maxx-1)-(3/4*maxx+5))/maxdcc;
setcolor( color );
if (wis) then
   begin
   bar(round(3/4*maxx+1),11,maxx-1,round(3/8*maxy-1));
   line(round(3/4*maxx+5),round(3/16*maxy),maxx-1,round(3/16*maxy));
   end;
for i:=1 to maxdcc do
    begin
    line(round(3/4*maxx+5+factor*(i-1)),round(3/16*maxy+ipchar[i-1]/maxint*3/32*maxy),
         round(3/4*maxx+5+factor*(i  )),round(3/16*maxy+ipchar[i  ]/maxint*3/32*maxy));
    end;
end;

procedure showfour( ipchar : fmask; color : integer; wis : boolean);
var  i : integer;
factor : real;
begin
factor:=((maxx-1)-(3/4*maxx+5))/maxdccshr;
setcolor( color );
if (wis) then
   begin
   bar(round(3/4*maxx+1),round(3/8*maxy+11),maxx-1,round(3/4*maxy-1));
   line(round(3/4*maxx+5),round((3/8+3/16)*maxy),maxx-1,round((3/8+3/16)*maxy));
   end;
for i:=1 to maxdccshr do
    begin
    line(round(3/4*maxx+5+factor*(i-1)),round((3/8+3/16)*maxy-ipchar[i-1,1]*1/64*maxy),
         round(3/4*maxx+5+factor*(i  )),round((3/8+3/16)*maxy-ipchar[i   ,1]*1/64*maxy));
    end;
end;

procedure displaychar( ipchar : mask; color : integer; wis : boolean );
var i,xp,yp    : integer;
    angle      : real;
const radi      = 7;
begin
angle:=0;
if wis then
   begin
   setfillstyle(1,0);
   bar(1,1,round(maxx*3/4-1),round(maxy*3/4-1));
   setfillstyle(1,15);
   bar(1,1,round(maxx*3/8),10);
   setfillstyle(1,0);
   setcolor(4);
   outtextxy(2, 2, ' Drawn Character:');
   end;
setcolor(color);
xp:=round(maxx/8*3);  yp:=round(maxy*3/8);
for i:=0 to maxdcc do
   begin
   angle:=angle-ipchar[i]/maxint*pi;
   line(xp,yp,xp+round(radi*cos(angle)),round(yp+radi*(sin(angle))));
   xp:=xp+round(radi*cos(angle));  yp:=yp+round(radi*sin(angle));
   end;
end;

function matchfour                 : integer;
var i,j,memi                       : integer;
res,min: real;
begin
min:=9999;     memi:=32;
assign(mcfile,'a:\chrdat\masks.dat');
reset(mcfile);
read(mcfile,memchars);
for i:=32 to 127 do
begin
   res:=0;
   for j:=1 to maxdccshr do
      begin
      res:=res+(memfour[j,1]-memchars.dcc.fouriermask[i,j,1])*(memfour[j,1]-memchars.dcc.f
   end;
   if res<min then
   begin
      memi:=i;
      min:=res;
      end;           { end element number }
   end;               { end character      }
   close(mcfile);
   matchfour:=memi;
end;                       { end of function }



function matchch( maxim : integer) : integer;
var i,j,memi,offset,tel    : integer;
maxok,ok,tussen,
totussen        : real;
begin
memi:=0;maxok:=0;
assign(mcfile,'a:\chrdat\masks.dat');
reset(mcfile);
read(mcfile,memchars);
for i:=32 to 127 do
begin
   if (maxim>0)                    and (memchars.dcc.maxarray[i]/maxim < 3/2) and
   (memchars.dcc.maxarray[i]>0) and (maxim/memchars.dcc.maxarray[i] > 2/3) then
   begin
      for offset:=-1 to 1 do
      begin
         ok:=0;      totussen:=0;
         for tel:=0 to maxdcc do
         begin
            tussen:=0;
            j:=tel+offset * 2 ;
            if (j>0) and (j<=maxdcc)  then tussen:=tussen+memchars.dcc.maskarray[i,j-1];
            if (j>=0) and (j<maxdcc)  then tussen:=tussen+memchars.dcc.maskarray[i,j+1];
            if (j>=0) and (j<=maxdcc) then tussen:=tussen+memchars.dcc.maskarray[i,j  ];
            if (j>0) and (j<maxdcc) then tussen:=tussen/3 else
                       tussen:=tussen/2;
            if (j>0) and (j<maxdcc) then
            begin
                       totussen:=totussen+abs(tussen)*abs(memchar[j]);
                       ok:=ok+abs( sin((tussen-memchar[j])/maxint*pi/2 ));
    {    writeln(lst,j:10,ok:10:2,totussen:10:2,(tussen-memchar[j]):10:2);}
                       end;
                   end;
               if totussen>0 then
                   ok:=totussen/ok
               else ok:=-1;
               if maxok<ok then
                   begin
                   maxok:=ok;
                   memi:=i;
                   end;
               end;
           end;
       end;                                { end character   }
       close(mcfile);
matchch:=memi;
end;                                       { end function }

function getch: integer;
begin
readfromtablet;
if (ipbcounter>5) then     { iets geschreven }
   begin
   makestring2(inputbufferpointer,ipbcounter,str1bufferpointer,str1counter);
   no_updn2(str1bufferpointer,str1counter,str2bufferpointer,str2counter);
   getch:=makenatpar(str2bufferpointer,str2counter,memchar);
   fourier(memchar,memfour);
   showchar(memchar,7,true);
   showfour(memfour,7,true);
   displaychar(memchar,7,true);
   end;
```

```
end;

procedure main;
var i,max: integer;
begin
resstr:=''; fresstr:='';
setgraphmode(getgraphmode);
drawborder;          setfillstyle(solidfill,white);
line(0,              round(3/4*maxy),maxx,          round(3/4*maxy));
line(round(3/4*maxx),0,              round(3/4*maxx),round(3/4*maxy));
line(round(3/4*maxx),round(3/8*maxy),maxx,          round(3/8*maxy));
bar( 0              ,0,              round(3/8*maxx),10             );
bar( round(3/4*maxx),0,              maxx,          10              );
bar( round(3/4*maxx),round(3/8*maxy),maxx,          round(3/8*maxy+10));
bar( 0,              maxy-25,         maxx,          maxy            );
bar( 0,              round(3/4*maxy),round(3/8*maxx),round(3/4*maxy+10));
bar( round(3/4*maxx),round(3/4*maxy),maxx,          round(3/4*maxy+10));
setcolor(4);
outtextxy(round(maxx*3/4),2,               ' Nat.Parm.Spec. ');
outtextxy(round(maxx*3/4),round(maxy*3/8)+2,' Fourier Descriptors');
outtextxy(round(maxx*3/4),round(maxy*3/4)+2,' General parameters');
outtextxy(2,              maxy-23,         ' COMMANDS :');
outtextxy(2,              round(maxy*3/4)+2,' Interpreted Characters:');
setcolor(8);
outtextxy(2,              maxy-13,
     '   C = change character definitions, T = change wait time,  Q = Quit');
showmaxtime;
setfillstyle(1,0);
repeat
   ch:=#0;
   max:=getch;
if (ipbcounter>5) then
       begin
       i:=matchch(max);
       resstr:=resstr+char(i);
       outtextxy(2,round(3/4*maxy+30),resstr);
       memchar:=memchars.dcc.maskarray[i];
       displaychar(memchar,9,false);
       showchar(memchars.dcc.maskarray[i],9,false);
       i:=matchfour;
       fresstr:=fresstr+char(i);
       outtextxy(2,round(3/4*maxy+40),fresstr);
       showfour(memchars.dcc.fouriermask[i],9,false);
       sound(3000); delay(100); nosound;
       end;
   if keypressed then
       begin
       ch:=upcase(readkey);
       case ch of
       'C':begin
           setfillstyle(1,15); bar(0,maxy-15,maxx,maxy);
           outtextxy(2,maxy-10,' Enter character to update: ');
           ch:=readkey;  outtextxy(maxx shr 1,maxy-10,ch);
           setfillstyle(1,0);
           repeat
               max:=getch;
           until (ipbcounter>5) or keypressed;
           if (ipbcounter>5) then
               begin
               memchars.dcc.maskarray[ord(ch)]:=memchar;
               memchars.dcc.maxarray[ord(ch)]:=max;
               memchars.dcc.fouriermask[ord(ch)]:=memfour;
               end;
           ch:='1';
           setfillstyle(1,white);
           bar(1,maxy-25,maxx-1,maxy-1);
           outtextxy(22,             maxy-13,
             '   C = change character definitions, T = change wait time,  Q = Quit');
           setfillstyle(1,0);
           assign(mcfile,'a:\chrdat\masks.dat');
           rewrite(mcfile);
           write(mcfile,memchars);
           close(mcfile);
           end;
       'T': begin
```

```
          changetime;
          setfillstyle(1,white);
          bar(1,maxy-25,maxx-1,maxy-1);
          outtextxy(22,               maxy-13,
            '   C = change character definitions, T = change wait time,  Q = Quit');
          setfillstyle(1,0);
          end;
      'Q': ;
      ESC:begin
          ch:=readkey;
          assign(mcfile,'a:\chrdat\masks.dat');
          reset(mcfile);
          read(mcfile,memchars);
          showchar(memchars.dcc.maskarray[ord(ch)],10,true);
          showfour(memchars.dcc.fouriermask[ord(ch)],10,true);
          displaychar(memchars.dcc.maskarray[ord(ch)],10,true);
          delay(4000);
          close(mcfile);
          end;
      else beep;
      end;
   end;
until ch='Q';
end;


begin

overnieuw:

initialisation;                        { initiate variables }
allocate_buffers;                      { allocate dynamic buffers }
init_buffers;                          { clear buffers }

assign(mcfile,'a:\chrdat\masks.dat');
{$I-} reset(mcfile);{$I+}
if ioresult=0 then read(mcfile,memchars)
else begin
   rewrite(mcfile); beep; beep;
   fillchar(memchars,sizeof(charrec),0);
   write(mcfile,memchars);
   end;
close(mcfile);
main;


restorevar;                            { restore interrupt vectors }


closegraph;                            { shut of graphic system }
clrscr;



end.
```

2 : **Match2.pas.**

This program performs several functions in character processing :

- calculate ϕ and $\iota\phi$-functions from the X-Y-coordinate pairs.
- make partitioning from $\iota\phi$-function.
- perform straight line-matching, including Singular Value Decomposition.
- perform circle matching, using the SIMPLEX-optimizing numerical procedure.

```
{$A-,B-,D+,E+,F+,I+,L+,N+,O+,R+,S+,V-}
{$M 64000,0,655360}

program match2;

{$N+,I+,V+}


Uses Crt, Graph, printer;

Const x_screen   = 300 ;
      y_screen   = 75 ;
      peaklevel  = 2.2 ;
      bendlevel  = 0.5 ;



Type matmn = array[1..150,1..3] of real;
     matnn = array[1..3,1..3] of real;
     vecn  = array[1..3] of real;

     simpmat        = array[1..4,1..3] of double;
     vector         = array[1..3] of double;

     curve_rec      =       record
                                 curve_kind    : char;
                                 start_angle   : real;
                                 end_angle     : real;
                                 start_point_x : real;
                                 start_point_y : real;
                                 end_point_x   : real;
                                 end_point_y   : real;
                                 radius        : real;
                                 orientation   : boolean;
                            end;

     curve_array    =       array[1..11] of curve_rec;    { 11= max. # of segments}


Var i, GraphDriver, GraphMode, ErrorCode, Color,
    Pattern, numpoints, nsections, file_number    : integer;

    withu, withv, sign                            : boolean;
    c, f, g, h, s, x, y,
    help, z                                       : real;
    a, a_star, u                                  : matmn;
    q, e, house_v,
    house_w, result_vector                        : vecn;
    v , house                                     : matnn;
    j, k, l, l1, numdif,
    m, n                                          : integer;
    data_file, record_file                        : text;



    x4dat, x3dat, x2dat, x1dat,
    y4dat, y3dat, y2dat, y1dat, x2y2dat, x1y2dat,
    x2y1dat, x1y1dat, estim_radius1, estim_radius2,
    estim_radius3, estim_radius4, mx_init1, my_init1,
    mx_init2, my_init2, mx_init3, my_init3,
    mx_init4, my_init4, start_x,
    end_x, start_y, end_y                          : double;
    cir_array, total_array                         : array[1..100,1..2] of double;
    start_arc, end_arc                             : real ;

    image_size                                     : word;
    image_pointer                                  : pointer;

    curve_data                                     : curve_array;

(*
***************************************************************************
***************    END OF GLOBAL PARAMETER SPECIFICATIONS    ***************
***************************************************************************
*)
```

```
procedure show_char;
var x, y : real;

begin
   assign(data_file,'a:\chrdat\xy.dat');
   reset(data_file);
   GraphDriver:=Detect;
   DetectGraph(Graphdriver, GraphMode);
   InitGraph(GraphDriver, GraphMode,'a:\werk');
   ErrorCode:=GraphResult;
   if ErrorCode <> grOK then
   begin
      Writeln('Graphics error   ');
      writeln(GraphErrorMsg(ErrorCode));
      writeln('Program aborted  ');
      delay(5000);
      Halt(1);
   end;
   clrscr;
   Color:=15;
   SetColor(Color);
   SetFillStyle(Pattern, Color);
   Bar(0,0,619,399);
   Color:=0;
   SetColor(Color);
   SetFillStyle(Pattern, Color);
   i:=0;
   while NOT (i=numpoints) do
   begin
      i:=i+1;
      putpixel(round(x_screen + total_array[i,1]), y_screen - round(total_array[i,2]), 0);
      repeat;
      until readkey=' ';
   end;
   repeat;
   until readkey='q';
   Closegraph;
end;





procedure make_angles;
var tf, tg, th                          : text;
    i                                   : integer;
    x_i, y_i, theta, theta_prev         : real;

begin
   assign(tf,'a:\chrdat\xy.dat');
   assign(tg,'a:\chrdat\phi.dat');
   assign(th,'a:\chrdat\delphi.dat');
   reset(tf);
   rewrite(tg);
   rewrite(th);
   i:=0;
   repeat
      i:=i+1;
      readln(tf, x_i, y_i);
      total_array[i,1]:=x_i;
      total_array[i,2]:=y_i;
      numpoints:=i;            { not angles but # (x,y)-coordinates  }

      if i >= 2 then
      begin
         if total_array[i,1]= total_array[i-1,1] then
         begin
            if total_array[i,2] > total_array[i-1,2] then theta:=pi/2 else theta:=-pi/2;
         end
         else
```

```
        begin
           theta:=arctan((total_array[i,2] - total_array[i-1,2])/(total_array[i,1] - tota
           if total_array[i,1] < total_array[i-1,1] then theta:=theta + pi;
        end;

        if (theta - theta_prev)>pi then
           repeat
              theta:=theta- 2*pi;
           until (theta - theta_prev<=pi) and (theta - theta_prev>=-pi);
        if (theta - theta_prev)<-pi then
           repeat
              theta:=theta + 2*pi;
           until (theta - theta_prev<=pi) and (theta - theta_prev>=-pi);

        writeln(tg,theta);
        if i > 2 then writeln(th,abs(theta - theta_prev));
        theta_prev:=theta;
     end;
   until eof(tf);

   close(tf);
   close(tg);
   close(th);


end;                           {    of procedure make_angles    }



procedure swap(var a , b : real);
var swap_help : real;
begin
   swap_help:=a;
   a:=b;
   b:=swap_help;
end;


procedure swap_int(var c, d : integer);
var swap_help : integer;
begin
   swap_help:=c;
   c:=d;
   d:=swap_help;
end;



procedure partitio(var nsections : integer);


var dat_phi, dat_xy, dat_out         : text;
    i, j, numpoints, numsections,
    last_assignment, distance        : integer;
    peak_location                    : array[1..5] of integer;
    peak                             : array[1..5] of real;
    mean, variance, x_co, y_co       : real;
    item                             : array[1..100] of real;



begin
   clrscr;
   assign(dat_phi,'a:\chrdat\delphi.dat');
   reset(dat_phi);
   last_assignment:=0;
   distance:=3;          { distance - 1 = minimal length of a section }
   numpoints:=0;
   numsections:=0;
   for i:=1 to 5 do
   begin
      peak[i]:=0;
      peak_location[i]:=0;
   end;
   mean:=0;
   variance:=0;
   for i:=1 to 100 do item[i]:=0;
   while NOT eof(dat_phi) do
   begin
      numpoints:=numpoints+1;
      readln(dat_phi,item[numpoints]);
      if numpoints=1 then
      begin
         peak[1]:=item[numpoints];
         peak_location[1]:=numpoints;
         last_assignment:=numpoints;
         mean:=mean + item[numpoints];
      end;
      if numpoints=2 then
      begin
         peak[2]:=item[numpoints];
         peak_location[2]:=numpoints;
         last_assignment:=numpoints;
         if peak[2] > peak[1] then
         begin
            swap(peak[1], peak[2]);
            swap_int(peak_location[1], peak_location[2]);
         end;
         mean:=mean + item[numpoints];
      end;
      if numpoints=3 then
      begin
         peak[3]:=item[numpoints];
         peak_location[3]:=numpoints;
         last_assignment:=numpoints;
         if peak[3] > peak[2] then
         begin
            swap(peak[2], peak[3]);
            swap_int(peak_location[3], peak_location[2]);
         end;
         if peak[2] > peak[1] then
         begin
            swap(peak[1], peak[2]);
            swap_int(peak_location[1], peak_location[2]);
         end;
         mean:=mean + item[numpoints];
      end;
      if numpoints = 4 then
      begin
         peak[4]:=item[numpoints];
         peak_location[4]:=numpoints;
         last_assignment:=numpoints;
         if peak[4] > peak[3] then
         begin
            swap(peak[4], peak[3]);
            swap_int(peak_location[4], peak_location[3]);
         end;
         if peak[3] > peak[2] then
         begin
            swap(peak[3], peak[2]);
            swap_int(peak_location[3], peak_location[2]);
         end;
         if peak[2] > peak[1] then
         begin
            swap(peak[2], peak[1]);
            swap_int(peak_location[2], peak_location[1]);
         end;
         mean:=mean + item[numpoints];
      end;
      if numpoints=5 then
      begin
         peak[5]:=item[numpoints];
         peak_location[5]:=numpoints;
         last_assignment:=numpoints;
         if peak[5] > peak[4] then
         begin
            swap(peak[5], peak[4]);
            swap_int(peak_location[5], peak_location[4]);
      end;
      if peak[4] > peak[3] then
      begin
         swap(peak[4], peak[3]);
         swap_int(peak_location[4], peak_location[3]);
      end;
      if peak[3] > peak[2] then
      begin
         swap(peak[3], peak[2]);
         swap_int(peak_location[3], peak_location[2]);
      end;
      if peak[2] > peak[1] then
      begin
         swap(peak[2], peak[1]);
         swap_int(peak_location[2], peak_location[1]);
      end;
      mean:=mean + item[numpoints];
   end;
   if numpoints > 5 then
   begin
      if (item[numpoints] > peak[1]) then
      begin
         if abs(last_assignment-numpoints) < distance then
         begin
            if last_assignment=peak_location[1] then
            begin
               last_assignment:=numpoints;
               peak[1]:=item[numpoints];
               peak_location[1]:=numpoints;
            end
            else
            begin
               if last_assignment=peak_location[2] then
               begin
                  peak[2]:=peak[1];
                  peak_location[2]:=peak_location[1];
                  peak[1]:=item[numpoints];
                  peak_location[1]:=numpoints;
                  last_assignment:=numpoints;
               end
               else
               begin
                  if last_assignment=peak_location[3] then
                  begin
                     peak[3]:=peak[2];
                     peak_location[3]:=peak_location[2];
                     peak[2]:=peak[1];
                     peak_location[2]:=peak_location[1];
                     peak[1]:=item[numpoints];
                     peak_location[1]:=numpoints;
                     last_assignment:=numpoints;
                  end
                  else
                  begin
                     if last_assignment=peak_location[4] then
                     begin
                        peak[4]:=peak[3];
                        peak_location[4]:=peak_location[3];
                        peak[3]:=peak[2];
                        peak_location[3]:=peak_location[2];
                        peak[2]:=peak[1];
                        peak_location[2]:=peak_location[1];
                        peak[1]:=item[numpoints];
                        peak_location[1]:=numpoints;
                        last_assignment:=numpoints;
                     end
                     else
                     begin
                        if last_assignment=peak_location[5] then
                        begin
                           peak[5]:=peak[4];
                           peak_location[5]:=peak_location[4];
                           peak[4]:=peak[3];
                           peak_location[4]:=peak_location[3];
                           peak[3]:=peak[2];
                        peak_location[3]:=peak_location[2];
                        peak[2]:=peak[1];
                        peak_location[2]:=peak_location[1];
                        peak[1]:=item[numpoints];
                        peak_location[1]:=numpoints;
                        last_assignment:=numpoints;
                     end;
                  end;
               end;
            end;
         end;
      end
      else
      begin
         peak[5]:=peak[4];
         peak_location[5]:=peak_location[4];
         peak[4]:=peak[3];
         peak_location[4]:=peak_location[3];
         peak[3]:=peak[2];
         peak_location[3]:=peak_location[2];
         peak[2]:=peak[1];
         peak_location[2]:=peak_location[1];
         peak[1]:=item[numpoints];
         peak_location[1]:=numpoints;
         last_assignment:=numpoints;
      end;
   end
   else
   begin
      if item[numpoints]>peak[2] then
      begin
         if abs(last_assignment-numpoints)<distance then
         begin
            if last_assignment=peak_location[2] then
            begin
               peak[2]:=item[numpoints];
               peak_location[2]:=numpoints;
               last_assignment:=numpoints;
            end
            else
            begin
               if last_assignment=peak_location[3] then
               begin
                  peak[3]:=peak[2];
                  peak_location[3]:=peak_location[2];
                  peak[2]:=item[numpoints];
                  peak_location[2]:=numpoints;
                  last_assignment:=numpoints;
               end
               else
               begin
                  if last_assignment=peak_location[4] then
                  begin
                     peak[4]:=peak[3];
                     peak_location[4]:=peak_location[3];
                     peak[3]:=peak[2];
                     peak_location[3]:=peak_location[2];
                     peak[2]:=item[numpoints];
                     peak_location[2]:=numpoints;
                     last_assignment:=numpoints;
                  end
                  else
                  begin
                     if last_assignment=peak_location[5] then
                     begin
                        peak[5]:=peak[4];
                        peak_location[5]:=peak_location[4];
                        peak[4]:=peak[3];
                        peak_location[4]:=peak_location[3];
                        peak[3]:=peak[2];
                        peak_location[3]:=peak_location[2];
                        peak[2]:=item[numpoints];
                        peak_location[2]:=numpoints;
                        last_assignment:=numpoints;
                     end;
```

```
                end;
             end;
          end;
       end
       else
       begin
          peak[5]:=peak[4];
          peak_location[5]:=peak_location[4];
          peak[4]:=peak[3];
          peak_location[4]:=peak_location[3];
          peak[3]:=peak[2];
          peak_location[3]:=peak_location[2];
          peak[2]:=item[numpoints];
          peak_location[2]:=numpoints;
          last_assignment:=numpoints;
       end;
    end
    else
    begin
       if item[numpoints]>peak[3] then
       begin
          if abs(last_assignment-numpoints)<distance then
          begin
             if last_assignment=peak_location[3] then
             begin
                peak[3]:=item[numpoints];
                peak_location[3]:=numpoints;
                last_assignment:=numpoints;
             end
             else
             begin
                if last_assignment=peak_location[4] then
                begin
                   peak[4]:=peak[3];
                   peak_location[4]:=peak_location[3];
                   peak[3]:=item[numpoints];
                   peak_location[3]:=numpoints;
                   last_assignment:=numpoints;
                end
                else
                begin
                   if last_assignment=peak_location[5] then
                   begin
                      peak[5]:=peak[4];
                      peak_location[5]:=peak_location[4];
                      peak[4]:=peak[3];
                      peak_location[4]:=peak_location[3];
                      peak[3]:=item[numpoints];
                      peak_location[3]:=numpoints;
                      last_assignment:=numpoints;
                   end;
                end;
             end;
          end
          else
          begin
             peak[5]:=peak[4];
             peak_location[5]:=peak_location[4];
             peak[4]:=peak[3];
             peak_location[4]:=peak_location[3];
             peak[3]:=item[numpoints];
             peak_location[3]:=numpoints;
             last_assignment:=numpoints;
          end;
       end
       else
       begin
          if item[numpoints]>peak_location[4] then
          begin
             if abs(last_assignment-numpoints)<distance then
             begin
                if last_assignment=peak_location[4] then
                begin
                   peak[4]:=item[numpoints];
                   peak_location[4]:=numpoints;
```

```
                        last_assignment:=numpoints;
                     end
                     else
                     begin
                        if last_assignment=peak_location[5] then
                        begin
                           peak[5]:=peak[4];
                           peak_location[5]:=peak_location[4];
                           peak[4]:=item[numpoints];
                           peak_location[4]:=numpoints;
                           last_assignment:=numpoints;
                        end;
                     end;
                  end
                  else
                  begin
                     peak[5]:=peak[4];
                     peak_location[5]:=peak_location[4];
                     peak[4]:=item[numpoints];
                     peak_location[4]:=numpoints;
                     last_assignment:=numpoints;
                  end;
               end
               else
               begin
                  if item[numpoints]>peak_location[5] then
                  begin
                     if abs(last_assignment-numpoints)<distance then
                     begin
                        if last_assignment=peak_location[5] then
                        begin
                           peak[5]:=item[numpoints];
                           peak_location[5]:=numpoints;
                           last_assignment:=numpoints;
                        end;
                     end
                     else
                     begin
                        peak[5]:=item[numpoints];
                        peak_location[5]:=numpoints;
                        last_assignment:=numpoints;
                     end;
                  end;
               end;
            end;
         end;
      end;
      mean:=mean + item[numpoints];
   end;
end;
close(dat_phi);
mean:=mean/numpoints;
for i:=1 to numpoints do variance:=sqr(item[i] - mean);
variance:=variance/(numpoints-1);
writeln('mean     =   ',mean);
writeln('variance =   ',variance);
writeln;
writeln;
writeln('peak1    =   ',peak[1],'     ',peak_location[1]);
writeln('peak2    =   ',peak[2],'     ',peak_location[2]);
writeln('peak3    =   ',peak[3],'     ',peak_location[3]);
writeln('peak4    =   ',peak[4],'     ',peak_location[4]);
writeln('peak5    =   ',peak[5],'     ',peak_location[5]);
repeat;
until Keypressed ;

numsections:=6;
if (( peak[1]/mean < peaklevel)  OR (peak_location[1] < 4 )) then
begin
   numsections:=1;
   peak_location[1]:=-1;
   peak_location[2]:=-1;
   peak_location[3]:=-1;
   peak_location[4]:=-1;
   peak_location[5]:=-1;
```

```
end
else
begin
   if (( peak[2]/mean < peaklevel) OR ( peak_location[2] < 4 )) then
   begin
      numsections:=2;
      peak_location[2]:=-1;
      peak_location[3]:=-1;
      peak_location[4]:=-1;
      peak_location[5]:=-1;
   end
   else
   begin
      if (( peak[3]/mean < peaklevel ) OR (peak_location[3] < 4 )) then
      begin
         numsections:=3;
         peak_location[3]:=-1;
         peak_location[4]:=-1;
         peak_location[5]:=-1;
      end
      else
      begin
         if (( peak[4]/mean < peaklevel ) OR (peak_location[4] < 4 )) then
         begin
            numsections:=4;
            peak_location[4]:=-1;
            peak_location[5]:=-1;
         end
         else
         begin
            if (( peak[5]/mean < peaklevel ) OR (peak_location[5] < 4 )) then
            begin
               numsections:=5;
               peak_location[5]:=-1;
            end;
         end;
      end;
   end;
end;

nsections:=numsections;


writeln;
writeln('After deleting small peaks : ');
writeln('Number of real peaks =  ',(numsections-1));
writeln;
writeln('peak1    =    ',peak[1],'     ',peak_location[1]);
writeln('peak2    =    ',peak[2],'     ',peak_location[2]);
writeln('peak3    =    ',peak[3],'     ',peak_location[3]);
writeln('peak4    =    ',peak[4],'     ',peak_location[4]);
writeln('peak5    =    ',peak[5],'     ',peak_location[5]);
repeat;
until Keypressed ;

(*
Now partition into files, first deleting possible
existing files part1..part6.dat.
*)

for i:=1 to 6 do
begin
   assign(dat_out,'a:\chrdat\part'+chr(48+i)+'.dat');
   {$I-}
   reset(dat_out);
   {$I+}
   if IOResult = 0 then
   begin
      close(dat_out);
      erase(dat_out);
   end;
end;


assign(dat_xy,'a:\chrdat\xy.dat');
```

```
   reset(dat_xy);
   assign(dat_out,'a:\chrdat\part1.dat');
   rewrite(dat_out);
   i:=0;                       {     item number     }
   j:=1;                       {    sectionnumber    }
   while  NOT eof(dat_xy) do
   begin
      i:=i+1;
      readln(dat_xy,x_co,y_co);
      if NOT ((i=peak_location[1]+1) or (i=peak_location[2]+1) or
      (i=peak_location[3]+1) or (i=peak_location[4]+1) or
      (i=peak_location[5]+1)) then
      begin
         writeln(dat_out,x_co,'        ',y_co);
      end
      else
      begin
         close(dat_out);
         if j < nsections then
         begin
            j:=j+1;
            assign(dat_out,'a:\chrdat\part'+chr(48+j)+'.dat');
            {$I-} reset(dat_out); {$I+} if IOResult = 0 then
            begin
               close(dat_out);
               erase(dat_out);
            end
            else rewrite(dat_out);
         end;
      end;
   end;
   close(dat_xy);
   close(dat_out);
end;                          {   of procedure partitio  }




(*
*****************************************************************
************   END OF PARTITION PROCEDURES   ***************
*****************************************************************
*)




procedure test_f_conv;
begin
                    z:=q[k];
                    if l = k then
                    begin
                       if z<0 then
                       begin
                          {
                          q[k] is made non-negative
                          }
                          q[k]:=-z;
                          if withv then
                          begin
                             for j:=1 to n do v[j,k]:=-v[j,k];
                          end;
                       end;  {  z  }
                       l:=0;
                    end     {  l=k  } else
                    begin
                       {
                       Shift from bottom 2x2 minor
                       }
                       x:=q[l];
                       y:=q[k-1];
                       g:=e[k-1];
                       h:=e[k];
```

```
                      f:=((y-z)*(y+z) + (g-h)*(g+h))/(2*h*y);
                      g:=sqrt(f*f+1);
                      if f<0 then f:=((x-z)*(x+z) + h*(y/(f-g) + h))/x else
                                  f:=((x-z)*(x+z) + h*(y/(f+g) + h))/x;
                      {
                      Next QR transformations
                      }
                      s:=1;
                      c:=1;
                      for i:=l+1 to k do
                      begin
                         g:=e[i];
                         y:=q[i];
                         h:=s*g;
                         g:=c*g;
                         e[i-1]:=sqrt(f*f + h*h);
                         z:=e[i-1];
                         c:=f/z;
                         s:=h/z;
                         f:=x*c + g*s;
                         g:=-x*s + g*c;
                         h:=y*s;
                         y:=y*c;
                         if withv then
                         begin
                            for j:=1 to n do
                            begin
                               x:=v[j,i-1];
                               z:=v[j,i];
                               v[j,i-1]:=x*c + z*s;
                               v[j,i]:= -x*s + z*c;
                            end; { j }
                         end;   { withv }
                         q[i-1]:=sqrt(f*f + h*h);
                         z:=q[i-1];
                         c:=f/z;
                         s:=h/z;
                         f:=c*g + s*y;
                         x:=-s*g + c*y;
                         if withu then
                         begin
                            for j:=1 to m do
                            begin
                               y:=u[j,i-1];
                               z:=u[j,i];
                               u[j,i-1]:=y*c + z*s;
                               u[j,i]:=-y*s + z*c;
                            end;  { j }
                         end;   { withu }
                      end;    { i }
                      e[l]:=0;
                      e[k]:=f;
                      q[k]:=x;
                      l:=k+1;
                    end;    {  l=k-else        }
end;                         { of procedure test_f_conv   }




procedure householder;
var lengthv, lengthw : real;

begin
   numdif:=1;
   for i:=n downto 2 do
   begin
      if q[i-1] < q[i] then
      begin
         help:=q[i-1];             {  help is here a help variable  }
         q[i-1]:=q[i];
         q[i]:=help;
         for j:=1 to n do
         begin
            help:=v[j,i];
            v[j,i]:=v[j,i-1];
            v[j,i-1]:=help;
         end;
      end;
      if q[i-1] = q[i] then numdif:=numdif + 1;
   end;
   numdif:=n - numdif;

   {
   Now the sing. vals are in order and so is the matrix V
   }

   if numdif <> (n-1) then
   begin
      lengthv:=0;
      lengthw:=0;
      for i:=(numdif+1) to n do
      begin
         lengthv:=lengthv + sqr(v[n,i]);
         if i <> n then lengthw:=lengthw;
         house_w[i-numdif]:=v[n,i];
         if i = n then
         begin
            house_w[i]:=house_w[i] + sqrt(lengthv);
            lengthw:=sqrt(lengthw + sqr(house_w[i]));
         end;
         for j:=(numdif+1) to n do
         begin
            house[i-numdif,j-numdif]:=v[n,i]*v[n,j];
         end;
      end;
      for i:= 1 to (n-numdif) do
      begin
         for j:=1 to (n-numdif) do
         begin
            house[i,j]:=-2*house[i,j]/lengthw;
            if j=i then house[i,j]:=house[i,j]+1;
         end;
      end;
      for i:=1 to n do
      begin
         for j:=(numdif+1) to n do
         begin
            result_vector[i]:=v[i,j]*house[(j-numdif), (n-numdif)];
         end;
      end;
      for i:=1 to (n-1) do result_vector[i]:=result_vector[i]/result_vector[n];
   end;
end;




procedure layout_svd;
var line_start_x, line_start_y,
    line_end_x, line_end_y : real;

begin

   (*
   This part first calculates the number of different
   singular values, and at the same time puts the
   singular values in decreasing order. So also the
   rows of V(transpose) must be put in the same order.
   *)

   clrscr;
   for i:=1 to m do
   begin
      for j:=1 to n do
      begin
```

```
         write(u[i,j]:8:3);
      end;
      writeln;
   end;
   writeln;writeln;
   for i:=1 to n do writeln(q[i]:8:3);
   writeln;writeln;
   for i:=1 to n do
   begin
      for j:=1 to n do
      begin
         write(v[j,i]:8:3);   { warning : this is V(transpose) and not V !!! }
      end;
      writeln;
   end;
   writeln;writeln;
   for i:=1 to m do
   begin
      for j:=1 to n do
      begin
         a_star[i,j]:=0;
         for k:=1 to n do
         begin
            a_star[i,j]:=a_star[i,j] + (q[k]*v[j,k]*u[i,k]);
         end;
         write(a_star[i,j]:8:3);
      end;
      writeln;
      if whereY>=23 then
      begin
         writeln('page (p)...');
         repeat;
         until readkey='p';
         clrscr;
      end;
   end;
   writeln;writeln;
   writeln(numdif);
   repeat;
   until readkey='q';
   if numdif=(n-1) then
   begin
      clrscr;
      writeln;
      writeln('numdif = ',numdif);
      writeln('If numdif = 2 then no repeated singular values. ');
      writeln('In this case computations can proceed.         ');
      writeln;
      writeln;
      writeln('results for section  ',file_number,'  :  ');
      writeln;
      for i:=1 to (n-1) do
      begin
         result_vector[i]:=-v[i,n]/v[n,n];
         writeln(result_vector[i]);
      end;
   end;
   writeln;
   writeln('Display graphics (g) or quit (q) ?   ');
   if readkey='g'then
   begin
      GraphDriver:=Detect;
      DetectGraph(Graphdriver, GraphMode);
      InitGraph(GraphDriver, GraphMode,'a:\werk');
      ErrorCode:=GraphResult;
      if ErrorCode <> grOK then
      begin
         writeln('Graphics error  ');
         writeln(GrapherrorMsg(ErrorCode));
         Writeln('Program aborted');
         delay(5000);
         Halt(1);
      end;
      clrscr;
      Color:=15;
      SetColor(Color);
      SetFillStyle(Pattern, Color);
      Bar(0,0,619,399);
      Color:=0;
      SetColor(Color);
      SetFillStyle(Pattern, Color);

      if file_number <> 1 then
      begin
         PutImage(70,0,image_pointer^,NormalPut);
         repeat;
         until readkey=' ';
      end
      else
      begin
         Image_size:=ImageSize(70,0,500,300);
         GetMem(image_pointer, image_size);
         GetImage(70,0,500,300,image_pointer^);
      end;

      SetLineStyle(0,0,NormWidth);
      for i:=1 to m do
      begin
         putpixel(round(x_screen + a[i,1]), y_screen - round(a[i,3]),0);
         repeat;
         until readkey=' ';
      end;
      if result_vector[1] <> 0 then
      begin
         line_start_x:=(a[1,n] + (1/result_vector[1])*a[1,1] - result_vector[2])/
                        (result_vector[1] + (1/result_vector[1]));
         line_start_y:=result_vector[1]*line_start_x + result_vector[2];
         line_end_x:=(a[m,n] + (1/result_vector[1])*a[m,1] - result_vector[2])/
                       (result_vector[1] + (1/result_vector[1]));
         line_end_y:=result_vector[1]*line_end_x + result_vector[2];
      end
      else
      begin
         line_start_x:=a[1,1];
         line_start_y:=result_vector[2];
         line_end_x  :=a[m,1];
         line_end_y  :=result_vector[2];
      end;
      curve_data[file_number].start_angle:=arctan((line_end_y-line_start_y)/
                                                  (line_end_x-line_start_x));
      if (( line_start_x > line_end_x ) OR ((line_end_y-line_start_y < 0 ) AND
                         (line_end_x-line_start_x < 0))) then
         curve_data[file_number].start_angle:=curve_data[file_number].start_angle + pi;
      curve_data[file_number].end_angle:=curve_data[file_number].start_angle;
      curve_data[file_number].radius:=sqrt(sqr(line_end_x-line_start_x) + sqr(line_end_y-l
      curve_data[file_number].start_point_x:=line_start_x;
      curve_data[file_number].start_point_y:=line_start_Y;
      curve_data[file_number].end_point_x:=line_end_x;
      curve_data[file_number].end_point_y:=line_end_y;
      curve_data[file_number].orientation:=FALSE;

      line(round(x_screen + line_start_x), y_screen - round(line_start_y),
         x_screen + round(line_end_x), round(y_screen - line_end_y));
      repeat;
      until readkey='q';



      image_size:=imagesize(70,0,500,300);
      GetImage(70,0,500,300,image_pointer^);


      CloseGraph;
   end;
end;                              { of procedure layout   }
```

```
procedure svd;


(*
******************************************************************************
*                                                                            *
*   This procedure calculates the singular value decompostion               *
*   of a matrix A(mxn) with m the number of rows and n the number           *
*   of columns. m>=n is assumed. The method used is a transcription of      *
*   the program of G.H. Golub and C. Reinsch in their article               *
*   "Singular Value Decomposition and Least Squares Solutons" in            *
*   Numer. Math. 14, pp.403-420 (1970).                                      *
*                                                                            *
******************************************************************************
*)


Var eps, tol : real;


begin
   curve_data[file_number].curve_kind:='l';
   assign(data_file,'a:\chrdat\part'+chr(48+file_number)+'.dat');
   reset(data_file);
   n:=3;
   m:=0;
   repeat
      m:=m+1;
      readln(data_file,a[m,1],a[m,3]);
      a[m,2]:=1;
   until eof(data_file);
   close(data_file);
   clrscr;
   withu:=true;
   withv:=true;
   eps:=1E-9;
   tol:=1E-10;
   g:=0;
   x:=0;
   for i:=1 to m do
   begin
      for j:=1 to n do u[i,j]:=a[i,j];
   end;
   {
   Householder reduction to bidiagonal form
   }
   g:=0;
   x:=0;
   for i:=1 to n do
   begin
      e[i]:=g;
      s:=0;
      l:=i + 1;
      for j:=i to m do s:=s + sqr(u[j,i]);
      if s<tol then g:=0 else
      begin
         f:=u[i,i];
         if f<0 then g:=sqrt(s) else g:=-sqrt(s);
         h:=f*g-s;
         u[i,i]:=f - g;
         for j:=l to n do
         begin
            if j<=n then
            begin
               s:=0;
               for k:=i to m do s:=s + u[k,i]*u[k,j];
               f:=s/h;
               for k:=i to m do u[k,j]:=u[k,j] + f*u[k,i];
            end;
         end;  { j  }
      end;     { s  }
      q[i]:=g;
      s:=0;
      for j:=l to n do
      begin
```

```
         if j <=n then s:=s + sqr(u[i,j]);
      end;
      if s<tol then g:=0 else
      begin
         f:=u[i,i+1];
         if f<0 then g:=sqrt(s) else g:=-sqrt(s);
         h:=f*g-s;
         u[i,i+1]:=f - g;
         for j:=l to n do
         begin
            if j <=n then e[j]:=u[i,j]/h;
         end;
         for j:=l to m do
         begin
            s:=0;
            for k:=l to n do
            begin
               if k <=n then s:=s + u[j,k]*u[i,k];
            end;
            for k:=l to n do
            begin
               if k <=n then u[j,k]:=u[j,k] + s*e[k];
            end;
         end;  {  j  }
      end;     {  s  }
      y:=abs(q[i]) + abs(e[i]);
      if y > x then x:=y;
   end;        {  i  }
   {
   Accumulation of right hand transformation
   }
   if withv then
   begin
      for i:=n downto 1 do
      begin
         if g <> 0 then
         begin
            h:=u[i,i+1]*g;
            for j:=l to n do
            begin
               if j <=n then v[j,i]:=u[i,j]/h;
            end;
            for j:=l to n do
            begin
               if j <=n then
               begin
                  s:=0;
                  for k:=l to n do
                  begin
                     if k <=n then s:=s + u[i,k]*v[k,j];
                  end;
                  for k:=l to n do
                  begin
                     if k <=n then v[k,j]:=v[k,j]+ s*v[k,i];
                  end;
               end;
            end;  {  j  }
         end;     {  g  }
         for j:=l to n do
         begin
            if j <=n then
            begin
               v[i,j]:=0;
               v[j,i]:=0;
            end;
         end;
         v[i,i]:=1;
         g:=e[i];
         l:=i;
      end;  {  i  }
   end;     { withv }
   {
   Accumulation of left hand transformations
   }
   if withu then
   begin
      for i:=n downto 1 do
      begin
         l:=i + 1;
         g:=q[i];
         for j:=l to n do
         begin
            if j <=n then u[i,j]:=0;
         end;
         if g<> 0 then
         begin
            h:=u[i,i]*g;
            for j:=l to n do
            begin
               if j<=n then
               begin
                  s:=0;
                  for k:=l to m do s:=s + u[k,i]*u[k,j];
                  f:=s/h;
                  for k:=i to m do u[k,j]:=u[k,j] + f*u[k,i];
               end;
            end;  {  j  }
            for j:=i to m do u[j,i]:=u[j,i]/g;
         end  {  g  }    else
         begin
            for j:=i to m do u[j,i]:=0;
         end;
         u[i,i]:=u[i,i]+1;
      end;   {  i  }
   end;     { withu }
   {
   Diagonalization of the bidiagonal form
   }
   eps:=eps*x;
   k:=n;
   while k>=1 do
   begin
      l:=k;
      while l>=1  do
      begin
         if abs(e[l])<=eps then
         begin
            test_f_conv;
         end else
         begin
            if abs(q[l-1]) <= eps then
            begin
               {
               Cancellation  of e[l] only if l >1 !
               }
               if l >1 then
               begin
                  c:=0;
                  s:=1;
                  l1:=l - 1;
                  i:=l;
                  while i <=k do
                  begin
                     f:=s*e[i];
                     e[i]:=c*e[i];
                     if abs(f) > eps then
                     begin
                        g:=q[i];
                        q[i]:=sqrt(f*f + g*g);
                        h:=q[i];
                        c:=g/h;
                        s:=-f/h;
                        if withu then
                        begin
                           for j:=1 to m do
                           begin
                              y:=u[j,l1];
                              z:=u[j,i];
                              u[j,l1]:=y*c + z*s;
                              u[j,i]:=-y*s + z*c;
```

```
                          end;  { j }
                       end;    { withu }
                    end;      { abs(f)  }
                    i:=i+1;
                 end;         { i=k downto 1-while  }
                 test_f_conv;
              end;            { if l > 1-condition  }
           end;            { abs(q[l-1])-if      }
        end;               { abs(e[l])-else      }
        l:=l-1;
     end;                  { l:=k downto 1-while }
     k:=k-1;
  end;                     { k:=n downto 1-while }
  {
  End of the SVD calculation part
  }


  householder;


  layout_svd;

end;              {   of procedure svd   }



(*
******************************************************************************
******************   END OF ALL THE SVD PROCEDURES   ********************
******************************************************************************
*)





procedure init_simp(filnum : integer; var num_points:integer;
                    var start_x, start_y, end_x, end_y : double);

var tf                                          : text;
    N                                           : integer;
    x_N, y_N, x_Nmin1,
    x_Nmin2, y_Nmin1, y_Nmin2,
    teller, noemer,
    lambda, delta, theta1, theta2,theta_prev    : double;

begin
   assign(tf,'a:\chrdat\part'+chr(48+filnum)+'.dat');
   {$I-} reset(tf); {$I+} if IOResult <> 0 then rewrite(tf);
   clrscr;
   N:=0;
   num_points:=0;
   x4dat:=0; x3dat:=0; x2dat:=0; x1dat:=0;
   y4dat:=0; y3dat:=0; y2dat:=0; y1dat:=0;
   x1y1dat:=0; x1y2dat:=0; x2y1dat:=0; x2y2dat:=0; estim_radius1:=0;
   estim_radius2:=0; estim_radius3:=0; estim_radius4:=0;
   writeln('    x-coordinate            y-coordinate  ');
   writeln('__________________________________________________________');
   writeln;
   repeat

      N:=N+1;

      readln(tf,x_N, y_N);                 { first read x-coordinate }
      cir_array[N,1]:=x_N;
      cir_array[N,2]:=y_N;
      writeln(x_N:7:3,'                       ', y_N:7:3);
      if whereY>=23 then
      begin
         writeln('page (p)...');
         repeat;
         until readkey='p';
   clrscr;
end;
x4dat:=x4dat + (x_N*x_N*x_N*x_N);
x3dat:=x3dat + (x_N*x_N*x_N);
x2dat:=x2dat + (x_N*x_N);
x1dat:=x1dat + x_N;

                                    { and then y-coordinate  }
y4dat:=y4dat + (y_N*y_N*y_N*y_N);
y3dat:=y3dat + (y_N*y_N*y_N);
y2dat:=y2dat + (y_N*y_N);
y1dat:=y1dat + y_N;

x2y2dat:=x2y2dat + (x_N*x_N*y_N*y_N);     {  cross terms   }
x2y1dat:=x2y1dat + (x_N*x_N*y_N);
x1y2dat:=x1y2dat + (x_N*y_N*y_N);
x1y1dat:=x1y1dat + (x_N*y_N);

if N = 1 then
begin
   x_Nmin2:=x_N;
   start_x:=x_N;
   y_Nmin2:=y_N;
   start_y:=y_N;
end;

if N = 2 then
begin
   x_Nmin1:=x_N;
   y_Nmin1:=y_N;
   if x_Nmin1 = x_Nmin2 then
   begin
      if y_Nmin1 > y_Nmin2 then theta2:=pi/1 else theta2:=-pi/2;
   end
   else
   begin
      theta2:=arctan((y_Nmin1 - y_Nmin2)/(x_Nmin1 - x_Nmin2));
      if x_Nmin1 < x_Nmin2 then theta2:=theta2 + pi;
   end;
   theta_prev:=theta2;

end;

if N>2 then
begin
   if x_N= x_Nmin1 then
   begin
      if y_N>y_Nmin1 then theta2:=pi/2 else theta2:=-pi/2;
   end
   else
   begin
      theta2:=arctan((y_N - y_Nmin1)/(x_N- x_Nmin1));
      if x_N<x_Nmin1 then theta2:=theta2 + pi;
   end;
   theta_prev:=theta2;

   if x_Nmin1 = x_Nmin2 then
   begin
      if y_Nmin1 > y_Nmin2 then theta1:=pi/2 else theta1:=-pi/2;
   end
   else
   begin
      theta1:=arctan((y_Nmin1 - y_Nmin2)/(x_Nmin1 - x_Nmin2));
      if x_Nmin1 < x_Nmin2 then theta1:=theta1 + pi;
   end;
   teller:=theta2 - theta1;

   noemer:=sqrt(((x_N - x_Nmin1)*(x_N - x_Nmin1)) +  ((y_N - y_Nmin1)*(y_N - y_Nmin1
   if noemer <> 0 then
   begin
      estim_radius1:=estim_radius1 + (teller/noemer);
   end;
   x_Nmin2:=x_Nmin1;
   y_Nmin2:=y_Nmin1;
   x_Nmin1:=x_N;
```

```
         y_Nmin1:=y_N;
      end;
      end_x:=x_N;
      end_y:=y_N;


   until eof(tf);
   close(tf);
   sign:=FALSE;
   if estim_radius1<0 then sign:=TRUE;
   if estim_radius1 <> 0 then estim_radius1:=(N-2)/abs(estim_radius1) else
   estim_radius1:=500;
   if estim_radius1 = 0 then estim_radius1:=1/10;
   delta:=estim_radius1/50;
   estim_radius2:=estim_radius1 + delta;
   estim_radius3:=estim_radius1;
   estim_radius4:=estim_radius1 + delta;
   if ( estim_radius1 >= ((y_N-start_y)*(y_N-start_y) + (x_N-start_x)*(x_N-start_x))/2 ) t
   begin
      lambda:=(estim_radius1*estim_radius1)-(((y_N - start_y)*(y_N - start_y) + (x_N - sta
      lambda:=sqrt(abs(lambda/((y_N - start_y)*(y_N - start_y) + (x_N - start_x)*(x_N - st
      if sign=FALSE then lambda:=-lambda;
   end
   else lambda:=0;

   mx_init1:=((start_x + X_N)/2) + (lambda*(y_N - start_y));
   my_init1:=((start_y + y_N)/2) - (lambda*(x_N - start_x));
   mx_init2:=mx_init1 + delta;
   my_init2:=my_init1;
   mx_init3:=mx_init1;
   my_init3:=my_init1 + delta;
   mx_init4:=mx_init1;
   my_init4:=my_init1;

   num_points:=N;
   writeln('busy optimizing...');

end;                                         {    end of procedure init_simp }







function error(mx,my,ra :double):double;
Var x4, x3, x2, x1,
    y4, y3, y2, y1,
    x1y1, x2y1, x1y2, x2y2, term                     : double;

begin

   x4:=(mx*mx*mx*mx);
   x3:=(mx*mx*mx);
   x2:=(mx*mx);
   x1:=(mx);

   y4:=(my*my*my*my);
   y3:=(my*my*my);
   y2:=(my*my);
   y1:=(my);

   x1y1:=(mx*my);
   x1y2:=(mx*my*my);
   x2y1:=(mx*mx*my);
   x2y2:=(mx*mx*my*my);

   term:=0;
   term:=term + (numpoints*x4) + x3*(-4*x1dat) +
            x2*(6*x2dat + 2*y2dat - 2*numpoints*ra*ra) + x1*((-4*x3dat) - (4*x1y2dat) + (
   term:=term + (numpoints*y4) + y3*(-4*y1dat) +
            y2*((6*y2dat) + (2*x2dat) - (2*numpoints*ra*ra)) + y1*((-4*y3dat) - (4*x2y1da
   term:=term + (x2y2*2*numpoints) + x2y1*(-4*y1dat) + x1y2*(-4*x1dat) + x1y1*(8*x1y1dat);
   term:=term + x4dat + y4dat + 2*x2y2dat - 2*ra*ra*x2dat -  2*ra*ra*y2dat + ra*ra*ra*ra*n
   error:=term;

end;                                {   end of function   }




procedure simpcalc(var mx1, my1, r1, mx2, my2, r2, mx3, my3, r3, mx4, my4, r4 : double);



(*
*************************************************************
*                                                           *
*    For a treatment of the theory behind this method I     *
* refer to "optimization, theory and applications", by      *
* S.S.Rao.                                                  *
*                                                           *
*                                       copyright L.N.90    *
*                                                           *
*************************************************************
*)



var alpha, beta, gamma, Q ,
    fmin, fmax, f_reflex,
    f_expand, f_contract, eps, hulp,scale : double;
    simp, datvec                          : simpmat;
    centroid, M_r, M_e, M_c               : vector;
    i                                     : integer;
    quant                                 : array[1..4] of double;
    contract_requirement                  : boolean;

begin
   alpha:=1;
   beta:=0.5;
   gamma:=2;
   eps:=1E-4;
   simp[1,1]:=mx1;
   simp[1,2]:=my1;
   simp[1,3]:=r1;
   simp[2,1]:=mx2;
   simp[2,2]:=my2;
   simp[2,3]:=r2;
   simp[3,1]:=mx3;
   simp[3,2]:=my3;
   simp[3,3]:=r3;
   simp[4,1]:=mx4;
   simp[4,2]:=my4;
   simp[4,3]:=r4;

   repeat

      quant[1]:=error(simp[1,1], simp[1,2], simp[1,3]);
      quant[2]:=error(simp[2,1], simp[2,2], simp[2,3]);
      quant[3]:=error(simp[3,1], simp[3,2], simp[3,3]);
      quant[4]:=error(simp[4,1], simp[4,2], simp[4,3]);



      for i:=1 to 3 do
      begin
         if quant[i+1]<quant[i] then
         begin
            hulp:=quant[i+1];
            quant[i+1]:=quant[i];
            quant[i]:=hulp;
            hulp:=simp[i+1,1];
            simp[i+1,1]:=simp[i,1];
            simp[i,1]:=hulp;
         hulp:=simp[i+1,2];
         simp[i+1,2]:=simp[i,2];
         simp[i,2]:=hulp;
         hulp:=simp[i+1,3];
         simp[i+1,3]:=simp[i,3];
         simp[i,3]:=hulp;
      end;
   end;
   fmax:=quant[4];
   scale:=fmax;

   for i:=4 downto 2 do
   begin
      if quant[i] < quant[i-1] then
      begin
         hulp:=quant[i];
         quant[i]:=quant[i-1];
         quant[i-1]:=hulp;
         hulp:=simp[i,1];
         simp[i,1]:=simp[i-1,1];
         simp[i-1,1]:=hulp;
         hulp:=simp[i,2];
         simp[i,2]:=simp[i-1,2];
         simp[i-1,2]:=hulp;
         hulp:=simp[i,3];
         simp[i,3]:=simp[i-1,3];
         simp[i-1,3]:=hulp;
      end;
   end;
   fmin:=quant[1];


   {
   Now the matrix is sorted
   }


   centroid[1]:=0;
   centroid[2]:=0;
   centroid[3]:=0;
   for i:=1 to 4 do
   begin
      if i <> 4 then                 { max ( i=4 ) is excluded  }
      begin
         centroid[1]:=centroid[1] + (simp[i,1]/3);
         centroid[2]:=centroid[2] + (simp[i,2]/3);
         centroid[3]:=centroid[3] + (simp[i,3]/3);
      end;
   end;

   M_r[1]:=((1 + alpha )*centroid[1]) - (alpha*(simp[4,1]));
   M_r[2]:=((1 + alpha )*centroid[2]) - (alpha*(simp[4,2]));
   M_r[3]:=((1 + alpha )*centroid[3]) - (alpha*(simp[4,3]));

   f_reflex  :=error(M_r[1], M_r[2], M_r[3]);

   contract_requirement:=TRUE;
   if (f_reflex < fmin ) then
   begin
      M_e[1]:=((gamma*M_r[1]) + ((1 - gamma)*centroid[1]));
      M_e[2]:=((gamma*M_r[2]) + ((1 - gamma)*centroid[2]));
      M_e[3]:=((gamma*M_r[3]) + ((1 - gamma)*centroid[3]));
      f_expand:=error(M_e[1], M_e[2], M_e[3]);
      if (f_expand < fmin) then
      begin
         simp[4,1]:=M_e[1];
         simp[4,2]:=M_e[2];
         simp[4,3]:=M_e[3];
      end
      else
      begin
         simp[4,1]:=M_r[1];
         simp[4,2]:=M_r[2];
         simp[4,3]:=M_r[3];
      end;
      end
      else
      begin
         for i:=1 to 3 do            {  4 not included because the highest is excluded }
         begin
            if (f_reflex <= quant[i]) then contract_requirement:=FALSE;
         end;
         if contract_requirement= FALSE then
         begin
            simp[4,1]:=M_r[1];
            simp[4,2]:=M_r[2];
            simp[4,3]:=M_r[3];
         end
         else
         begin
            if (NOT(f_reflex > fmax )) then
            begin
               simp[4,1]:=M_r[1];
               simp[4,2]:=M_r[2];
               simp[4,3]:=M_r[3];
               fmax:=error(M_r[1],M_r[2],M_r[3]);
            end;
            M_c[1]:=beta*simp[4,1] + ((1 - beta)*centroid[1]);
            M_c[2]:=beta*simp[4,2] + ((1 - beta)*centroid[2]);
            M_c[3]:=beta*simp[4,3] + ((1 - beta)*centroid[3]);
            f_contract:=error(M_c[1], M_c[2], M_c[3]);

            if (NOT(f_contract > fmax )) then
            begin
               simp[4,1]:=M_c[1];
               simp[4,2]:=M_c[2];
               simp[4,3]:=M_c[3];
            end
            else
            begin
               for i:=1 to 4 do
               begin
                  simp[i,1]:=((simp[i,1] + simp[1,1])/2);
                  simp[i,2]:=((simp[i,2] + simp[1,2])/2);
                  simp[i,3]:=((simp[i,3] + simp[1,3])/2);
               end;
            end;
         end;
      end;
      Q:=0;
      for i:=1 to 4 do
      begin
         hulp:=error(simp[i,1], simp[i,2], simp[i,3]);
         hulp:=hulp - error(centroid[1], centroid[2], centroid[3]);
         hulp:=hulp*hulp;
         hulp:=hulp/4;
         Q:=Q+hulp;
      end;
      Q:=sqrt(Q)/scale;
   until (abs(Q) <=eps);

   mx1:=simp[1,1];  {centroid[1];}      {  these will be the export variables  }
   my1:=simp[1,2];  {centroid[2];}          {  of the procedure simpcalc           }
   r1 :=simp[1,3];  {centroid[3];}
   r2 :=Q;
   r3 :=eps;

end;                             {    end of procedure simpcalc }



(*
*****************************************************************************
********************      END OF SIMPLEX PROCEDURES      ********************
*****************************************************************************
*)
```

```
procedure match_main;
begin
   new(image_pointer);
   mark(image_pointer);
   Color:=0;
   Pattern:=1;
   nsections:=0;
   file_number:=1;
   for i:=1 to 100 do
   begin
      cir_array[i,1]:=0;
      cir_array[i,2]:=0;
      total_array[i,1]:=0;
      total_array[i,2]:=0;
   end;
   for i:=1 to 11 do
   begin
      curve_data[i].curve_kind    :='e';   { empty record  }
      curve_data[i].start_angle   :=0;
      curve_data[i].end_angle     :=0;
      curve_data[i].start_point_x :=0;
      curve_data[i].start_point_y :=0;
      curve_data[i].end_point_x   :=0;
      curve_data[i].end_point_y   :=0;
      curve_data[i].radius        :=-maxint;
   end;

   make_angles;
   show_char;
   partitio(nsections);
   for file_number:=1 to nsections do
   begin
      numpoints:=0;
      start_x:=0;
      end_x:=0;
      start_y:=0;
      end_y:=0;
      sign:=FALSE;

      init_simp(file_number, numpoints, start_x, start_y, end_x, end_y);
      if estim_radius1 < 300 then
      begin
         curve_data[file_number].curve_kind:='c';
         simpcalc(mx_init1, my_init1, estim_radius1,
                  mx_init2, my_init2, estim_radius2,
                  mx_init3, my_init3, estim_radius3,
                  mx_init4, my_init4, estim_radius4 );
         curve_data[file_number].radius:=estim_radius1;

         sound(440);delay(100);nosound;
         writeln('Display results in graphics (g) or in text (t)? ');
         writeln;writeln;
         repeat;
         until keypressed;
         repeat;
            if readkey='t' then
            begin
               writeln;
               writeln('___________________________________________________');
               writeln;
               writeln('         Calculation results of section ',file_number,':    ');
               writeln('___________________________________________________');
               writeln;
               writeln('Mx   = ',mx_init1:7:3,'       My = ',my_init1:7:3,'       r =  ',es
               writeln('Q    =  ',estim_radius2);
               writeln('eps =  ',estim_radius3);
            end
            else
            begin

               (*
               Now the circle and the data points will be displayed.
               *)
```

```
GraphDriver:=Detect;
DetectGraph(GraphDriver,GraphMode);
InitGraph(GraphDriver, GraphMode,'a:\werk');
Errorcode:=GraphResult;
if ErrorCode <> grOK then
begin
   Writeln('Graphics error  ');
   Writeln(GraphErrorMsg(ErrorCode));
   Writeln('Program aborted ');
   delay(5000);
   Halt(1);
end;
clrscr;
Color:=15;
SetColor(Color);
SetFillStyle(Pattern, Color);
Bar(0,0,619, 399);
Color:=0;
SetColor(Color);
SetFillStyle(Pattern, Color);


if file_number <> 1 then
begin
   PutImage(70,0,image_pointer^,NormalPut);
   repeat;
   until readkey=' ';
end
else
begin
   Image_size:=ImageSize(70,0,500,300);
   GetMem(image_pointer, image_size);
   GetImage(70,0,500,300,image_pointer^);
end;


SetLineStyle(0,0,NormWidth);
for i:=1 to numpoints do
begin
   putpixel(round(x_screen + cir_array[i,1]*1),y_screen - round(cir_array[i
   repeat;
   until readkey=' ';
end;

(*
Now calculating starting and ending angle of arc
*)

if cir_array[1,1] = mx_init1 then
begin
   if cir_array[1,2]>my_init1 then start_arc:=pi/2 else start_arc:=-pi/2;
end
else
begin
   start_arc:=arctan(abs((cir_array[1,2]-my_init1)/(cir_array[1,1]-mx_init1
   if cir_array[1,1]>mx_init1 then
   begin
      if cir_array[1,2]<my_init1 then start_arc:=-start_arc;
   end
   else
   begin
      if cir_array[1,2]<my_init1 then start_arc:=start_arc-pi
         else start_arc:=(-start_arc+pi);
   end;
end;
curve_data[file_number].start_angle:=start_arc;

start_arc:=(start_arc/pi)*180;                              { degrees }


if cir_array[numpoints,1] = mx_init1 then
begin
   if cir_array[numpoints,2]>my_init1 then end_arc:=pi/2 else end_arc:=-pi/
end
```

```
else
begin
   end_arc:=arctan(abs((cir_array[numpoints,2]-my_init1)/(cir_array[numpoin
   if cir_array[numpoints,1]>mx_init1 then
   begin
      if cir_array[numpoints,2]<my_init1 then end_arc:=-end_arc;
   end
   else
   begin
      if cir_array[numpoints,2]<my_init1 then end_arc:=end_arc-pi
         else end_arc:=(-end_arc+pi);
   end;
end;

curve_data[file_number].end_angle:=end_arc;

if ( curve_data[file_number].end_angle - curve_data[file_number].start_angl
      curve_data[file_number].end_angle:=curve_data[file_number].end_angle

if ( curve_data[file_number].start_angle - curve_data[file_number].end_angl
      curve_data[file_number].end_angle:=curve_data[file_number].end_angle

curve_data[file_number].start_point_x:=(estim_radius1*cos(
     curve_data[file_number].start_angle)) + (mx_init1);
curve_data[file_number].start_point_y:=(estim_radius1*sin(
     curve_data[file_number].start_angle)) + (my_init1);
curve_data[file_number].end_point_x:=(estim_radius1*cos(
     curve_data[file_number].end_angle)) + (mx_init1);
curve_data[file_number].end_point_y:=(estim_radius1*sin(
     curve_data[file_number].end_angle)) + (my_init1);
curve_data[file_number].orientation:=sign;



if curve_data[file_number].start_angle > curve_data[file_number].end_angle
begin
   curve_data[file_number].orientation:=TRUE;
   sign:=TRUE;
end
else
begin
   curve_data[file_number].orientation:=FALSE;
   sign:=FALSE;
end;



end_arc  :=(end_arc/pi)*180;                        { degrees }

if NOT (start_arc>=0)  then
begin
   repeat
      start_arc:=start_arc+360;
   until (start_arc>=0);
end;

if NOT (end_arc>=0)  then
begin
   repeat
      end_arc:=end_arc+360;
   until (end_arc>=0);
end;

if NOT (end_arc<=360)  then
begin
   repeat
      end_arc:=end_arc-360;
   until (end_arc<=360);
end;

if NOT (start_arc<=360)  then
begin
   repeat
      start_arc:=start_arc-360;
   until (start_arc<=360);
```

```
            end;


            if sign=TRUE then
            begin
               if start_arc > end_arc then
               begin
                  swap(start_arc, end_arc);

                  Arc(round(x_screen+mx_init1), y_screen-round(my_init1),
                  round(start_arc), round(end_arc), round(estim_radius1));
               end
               else
               begin

                  Arc(round(x_screen+mx_init1), y_screen-round(my_init1),
                  0, round(start_arc), round(estim_radius1));

                  Arc(round(x_screen+mx_init1), y_screen-round(my_init1),
                  round(end_arc), 360, round(estim_radius1));
               end;
            end
            else
            begin
               if start_arc > end_arc then
               begin

                  Arc(round(x_screen+mx_init1), y_screen-round(my_init1),
                  round(start_arc), 360, round(estim_radius1));

                  Arc(round(x_screen+mx_init1), y_screen-round(my_init1),
                  0, round(end_arc), round(estim_radius1));
               end
               else
               begin
                  Arc(round(x_screen+mx_init1), y_screen - round(my_init1),
                  round(start_arc), round(end_arc), round(estim_radius1));
               end;
            end;
            if sign=TRUE then
            begin
               curve_data[file_number].start_angle:=
                   curve_data[file_number].start_angle - (pi/2);
               curve_data[file_number].end_angle:=
                   curve_data[file_number].end_angle - (pi/2);
            end
            else
            begin
               curve_data[file_number].start_angle:=
                   curve_data[file_number].start_angle + (pi/2);
               curve_data[file_number].end_angle:=
                   curve_data[file_number].end_angle + (pi/2);
            end;

            image_size:=imagesize(70,0,500,300);
            getImage(70,0,500,300,image_pointer^);

            repeat;
            until readkey='e';

            CloseGraph;
         end;
      until readkey='q';
   end
   else
   begin
      clrscr;
      writeln('estimated radius > 300 so line approximation is better ');
      repeat;
      until KeyPressed;
      svd;
   end;
end;
assign(record_file,'a:\chrdat\curvrec.dat');
rewrite(record_file);
```

```
   for i:=1 to 11 do
   begin
      writeln(record_file,curve_data[i].curve_kind);
      writeln(record_file,curve_data[i].start_angle);
      writeln(record_file,curve_data[i].end_angle);
      writeln(record_file,curve_data[i].start_point_x);
      writeln(record_file,curve_data[i].start_point_y);
      writeln(record_file,curve_data[i].end_point_x);
      writeln(record_file,curve_data[i].end_point_y);
      writeln(record_file,curve_data[i].radius); { or length in case of a straight line }
      writeln(record_file,curve_data[i].orientation);
   end;
   close(record_file);
end;




{
==============================================================================

           END OF FUNCTIONS AND PROCEDURES, MAIN PROGRAM STARTS HERE

===============================================================================
}





begin
        match_main;
end.
```

3 : **Calcspec.pas.**

Here, computations of GFDs takes place. Since starting and final angles are known for each section, we can compute the GFDs, making use of eq. 2-7, 2-21a and 2-21b. The results are written to data-files A:\FOURIER\AMSPEC.DAT (amplitude spectrum) and A:\FOURIER\PHSPEC.DAT (phase spectrum).

```
{$A-,B-,D+,E+,F+,I+,L+,N+,O+,R-,S-,V-}
{$M 16384,0,655360}
program calcspec;


Uses Crt, graph;



Const max_pieces = 20;       {  max number of curve segments   }
      max_freq   = 20;

      x_screen   = 300;
      y_screen   = 50;

      bendfactor = 1.3 ;


Type spectrum   = array[1..max_pieces] of real;


Type curv_dat = record
            curve_kind    : char    ;
            start_angle   : real    ;
            end_angle     : real    ;
            start_point_x : real    ;
            start_point_y : real    ;
            end_point_x   : real    ;
            end_point_y   : real    ;
            radius        : real    ;
            orientation   : string  ;
       end;

       final_dat = record
               start_angle  : real    ;
               end_angle    : real    ;
               start_dist   : real    ;
               end_dist     : real    ;
               radius       : real    ;
               orientation  : boolean ;
         end;


       curv_array    = array[1..11] of curv_dat  ; { max. 11 # of segments }
       final_array   = array[0..20] of final_dat ;


var k, i, nsections,
    final_number                : integer     ;
    f, g, h                     : text        ;
    ch, antw                    : char        ;
    curv_rec                    : curv_array  ;
    final_rec, final_rec_star   : final_array ;



    ampl, phase                 : spectrum;
    a, b, f0                    : real;
    amdat, phdat                : text;





function tan(alpha : real) : real;
begin
   if cos(alpha) <> 0 then tan:=sin(alpha)/cos(alpha) else
   begin
      if sin(alpha) = 0 then tan:=0 else
      begin
         if sin(alpha) > 0 then tan:=1E6;
         if sin(alpha) < 0 then tan:=-1E6;
      end;
   end;
end;
```

```
procedure swap(var a , b : real);
var swap_help : real;
begin
   swap_help:=a;
   a:=b;
   b:=swap_help;
end;



procedure swap_int(var a , b : integer);
var swap_help : integer;
begin
   swap_help:=a;
   a:=b;
   b:=swap_help;
end;



function delta_distance(x1, y1, theta1, x2, y2, theta2 : real): real;

begin
   if (theta1 <> theta2) then delta_distance:=((sqrt(sqr(x2-x1) + sqr(y2-y1)))/2)*
                              abs(theta2 - theta1)
   else delta_distance:=sqrt(sqr(x2-x1) + sqr(y2-y1));
end;





procedure make_fills;
var k : integer;

begin
   assign(f,'a:\chrdat\curvrec.dat');
   assign(g,'a:\chrdat\final.dat');
   reset(f);
   rewrite(g);
   nsections:=0;    { number of sections of curv_rec (without insertions) }
   final_number:=0; { number of sections of final_rec (with insertions )  }

   for i:=0 to 20 do
   begin
      final_rec[i].start_angle:=0;
      final_rec[i].end_angle:=0;
      final_rec[i].start_dist:=0;
      final_rec[i].end_dist:=0;
      final_rec[i].radius:=0;
      final_rec[i].orientation:=FALSE;
   end;
   i:=0;

   while NOT eof(f) do
   begin
      i:=i+1;
      readln(f,curv_rec[i].curve_kind)   ;
      readln(f,curv_rec[i].start_angle)  ;
      readln(f,curv_rec[i].end_angle)    ;
      readln(f,curv_rec[i].start_point_x);
      readln(f,curv_rec[i].start_point_y);
      readln(f,curv_rec[i].end_point_x)  ;
      readln(f,curv_rec[i].end_point_y)  ;
      readln(f,curv_rec[i].radius)       ; { radius for circle , length for line }
      readln(f,curv_rec[i].orientation)  ;
      if curv_rec[i].curve_kind <> 'e' then nsections:=i;
   end;
```

```
for i:=1 to (nsections) do
begin
   final_number:=final_number + 1;
   final_rec[final_number].start_angle:=curv_rec[i].start_angle;
   final_rec[final_number].end_angle:=curv_rec[i].end_angle;

   if (final_rec[final_number].end_angle - final_rec[final_number].start_angle > (bendf
   begin
      repeat
         curv_rec[i].end_angle:=curv_rec[i].end_angle - (2*pi);
         final_rec[final_number].end_angle:=final_rec[final_number].end_angle-(2*pi);
         for k:=(i+1) to nsections do
         begin
            curv_rec[k].start_angle:=curv_rec[k].start_angle - (2*pi);
            curv_rec[k].end_angle:=curv_rec[k].end_angle - (2*pi);
         end;
      until (final_rec[final_number].end_angle - final_rec[final_number].start_angle <
   end;

   if (final_rec[final_number].start_angle - final_rec[final_number].end_angle > (bendf
   begin
      repeat
         curv_rec[i].end_angle:=curv_rec[i].end_angle + (2*pi);
         final_rec[final_number].end_angle:=final_rec[final_number].end_angle+(2*pi);
         for k:=(i+1) to nsections do
         begin
            curv_rec[k].start_angle:=curv_rec[k].start_angle + (2*pi);
            curv_rec[k].end_angle:=curv_rec[k].end_angle + (2*pi);
         end;
      until (final_rec[final_number].start_angle - final_rec[final_number].end_angle <
   end;

   final_rec[final_number].start_dist:=final_rec[final_number-1].end_dist ;
   final_rec[final_number].end_dist:=final_rec[final_number].start_dist +
      delta_distance(curv_rec[i].end_point_x, curv_rec[i].end_point_y,
                     curv_rec[i].end_angle, curv_rec[i].start_point_x,
                     curv_rec[i].start_point_y, curv_rec[i].start_angle);
   final_rec[final_number].radius:=curv_rec[i].radius;
   if curv_rec[i].orientation = 'FALSE' then
      final_rec[final_number].orientation:=FALSE else
      final_rec[final_number].orientation:=TRUE;

   if (NOT((curv_rec[i+1].start_point_x = curv_rec[i].end_point_x) AND
     (curv_rec[i+1].start_point_y = curv_rec[i].end_point_y)) AND
                  (    i <> nsections )) then
   begin
      final_number:=final_number + 1;
      final_rec[final_number].start_angle:=curv_rec[i].end_angle;
      final_rec[final_number].end_angle:=curv_rec[i+1].start_angle;

      if (final_rec[final_number].end_angle - final_rec[final_number].start_angle > (5*
      begin
         repeat
            final_rec[final_number].end_angle:=final_rec[final_number].end_angle-(2*pi)
            for k:=(i+1) to nsections do
            begin
               curv_rec[k].start_angle:=curv_rec[k].start_angle - (2*pi);
               curv_rec[k].end_angle:=curv_rec[k].end_angle - (2*pi);
            end;
         until (final_rec[final_number].end_angle - final_rec[final_number].start_angle
      end;

      if (final_rec[final_number].start_angle - final_rec[final_number].end_angle > (5*
      begin
         repeat
            final_rec[final_number].end_angle:=final_rec[final_number].end_angle+(2*pi)
            for k:=(i+1) to nsections do
            begin
               curv_rec[k].start_angle:=curv_rec[k].start_angle + (2*pi);
               curv_rec[k].end_angle:=curv_rec[k].end_angle + (2*pi);
            end;
         until (final_rec[final_number].start_angle - final_rec[final_number].end_angle
      end;
```

```
         final_rec[final_number].start_dist:=final_rec[final_number-1].end_dist;
         final_rec[final_number].end_dist:=final_rec[final_number].start_dist +
            delta_distance(curv_rec[i].end_point_x, curv_rec[i].end_point_y,
                           curv_rec[i].end_angle, curv_rec[i+1].start_point_x,
                           curv_rec[i+1].start_point_y, curv_rec[i+1].start_angle);

         final_rec[final_number].radius:=(sqrt(sqr(curv_rec[i].end_point_x -
                                                  curv_rec[i+1].start_point_x) +
                                              sqr(curv_rec[i].end_point_y -
                                                  curv_rec[i+1].start_point_y)))/2;

         if final_rec[final_number].end_angle < final_rec[final_number].start_angle then
            final_rec[final_number].orientation:=TRUE else
            final_rec[final_number].orientation:=FALSE ;
      end;
   end;
   for i:=1 to final_number do
   begin
      writeln(g,final_rec[i].start_angle);
      writeln(g,final_rec[i].end_angle);
      writeln(g,final_rec[i].start_dist);
      writeln(g,final_rec[i].end_dist);  { .radius not written because not    }
      writeln(g,final_rec[i].radius);
      writeln(g,final_rec[i].orientation);
   end;                                  { necessary for calculating spectrum }
   close(f);
   close(g);
end;


(*
****************************************************************************
****************************************************************************
****************************************************************************
*)



procedure make_phi_star;
var i                           : integer;
    ksi, total_length, offset : real;

begin
   assign(h,'a:\chrdat\dr_star.dat');
   rewrite(h);

   ksi:=(final_rec[final_number].end_angle-final_rec[1].start_angle)/(-2*pi);
   total_length:=final_rec[final_number].end_dist;



   for i:=1 to final_number do
   begin
      final_rec_star[i].start_dist:=final_rec[i].start_dist/total_length;
      final_rec_star[i].end_dist:=final_rec[i].end_dist/total_length;
      final_rec_star[i].start_angle:=final_rec[i].start_angle + (2*pi*ksi*
                                     final_rec_star[i].start_dist);
      final_rec_star[i].end_angle:=final_rec[i].end_angle + (2*pi*ksi*
                                   final_rec_star[i].end_dist);
   end;

   offset:=final_rec_star[1].start_angle;

   for i:=1 to final_number do
   begin
      final_rec_star[i].start_angle:=final_rec_star[i].start_angle - offset;
      final_rec_star[i].end_angle:=final_rec_star[i].end_angle - offset;

   end;



   for i:=1 to final_number do
   begin
```

```
      writeln(h,final_rec_star[i].start_dist,'       ',final_rec_star[i].start_angle);
      if i=final_number then writeln(h,final_rec_star[i].end_dist,'       ',
                                       final_rec_star[i].end_angle);
   end;


   close(h);
end;






function four_coef_a_star( l: integer ):real;

var i                          : integer;
    B, C, beta, gamma, previous : real;
begin
   previous:=0;
   for i:=1 to final_number do
   begin
      B:=final_rec_star[i].start_angle/(l*pi);
      C:=final_rec_star[i].end_angle/(l*pi);
      beta:=final_rec_star[i].start_dist*2*pi*l*f0;
      gamma:=final_rec_star[i].end_dist*2*pi*l*f0;
      previous:=previous + ((B-C)/(beta-gamma))*(cos(gamma)-cos(beta)) - B*sin(beta) + C*s
   end;
   four_coef_a_star:=previous;
end;




function four_coef_b_star( l: integer ): real;
var i                          : integer;
    B, C, beta, gamma, previous : real;
begin
   previous:=0;
   for i:=1 to final_number do
   begin
      B:=final_rec_star[i].start_angle/(l*pi);
      C:=final_rec_star[i].end_angle/(l*pi);
      beta:=final_rec_star[i].start_dist*2*pi*l*f0;
      gamma:=final_rec_star[i].end_dist*2*pi*l*f0;
      previous:=previous + ((B-C)/(beta-gamma))*(sin(beta)-sin(gamma)) - B*cos(beta) + C*c
   end;
   four_coef_b_star:=previous;
end;




procedure Draw_Arc(star,  ein : integer ; var xprev, yprev : real; radius : real; k: integ
var mx, my        : real;
    start, eind   : integer;

begin
   start:=star;
   eind:=ein;
   if final_rec[k-1].start_angle=final_rec[k-1].end_angle then
   begin
      if final_rec[k-1].end_angle < final_rec[k+1].start_angle then
      begin
         mx:=xprev - radius*cos(final_rec[k].start_angle - (pi/2));
         my:=yprev - radius*sin(final_rec[k].start_angle - (pi/2));
      end
      else
      begin
         mx:=xprev + radius*cos(final_rec[k].start_angle - (pi/2));
         my:=yprev + radius*sin(final_rec[k].start_angle - (pi/2));
      end;
   end

   else
   begin
      if final_rec[k].orientation=FALSE then
      begin
         mx:=xprev - radius*cos(final_rec[k].start_angle - (pi/2));
         my:=yprev - radius*sin(final_rec[k].start_angle - (pi/2));
      end
      else
      begin
         mx:=xprev + radius*cos(final_rec[k].start_angle - (pi/2));
         my:=yprev + radius*sin(final_rec[k].start_angle - (pi/2));
      end;
   end;




   if eind > 360 then
   begin
      repeat
         eind:=eind - 360;
      until eind <=360;
   end;

   if start > 360 then
   begin
      repeat
         start:=start - 360;
      until start <=360;
   end;

   if eind < 0 then
   begin
      repeat
         eind:=eind + 360;
      until eind >=0;
   end;

   if start < 0 then
   begin
      repeat
         start:=start + 360;
      until start >= 0;
   end;

   if start < eind then
   begin
      if final_rec[k].orientation=FALSE then Arc(x_screen + round(mx), y_screen-round(my),
      else
      begin
         Arc(x_screen + round(mx), y_screen-round(my), 0, start, round(radius));
         Arc(x_screen + round(mx), y_screen-round(my), eind, 360, round(radius));
      end;
   end
   else
   begin
      if final_rec[k].orientation=FALSE then
      begin
         Arc(x_screen + round(mx), y_screen-round(my), start, 360, round(radius));
         Arc(x_screen + round(mx), y_screen-round(my), 0, eind, round(radius));
      end
      else
      begin
         Arc(x_screen + round(mx), y_screen-round(my), eind, start, round(radius));
      end;
   end;
   { delay(2000); }
   if final_rec[k].orientation=FALSE then
   begin
      xprev:=(mx + final_rec[k].radius*cos(final_rec[k].end_angle - (pi/2)));
      yprev:=(my + final_rec[k].radius*sin(final_rec[k].end_angle - (pi/2)));
   end
   else
   begin
      xprev:=(mx + final_rec[k].radius*cos(final_rec[k].end_angle + (pi/2)));
      yprev:=(my + final_rec[k].radius*sin(final_rec[k].end_angle + (pi/2)));
   end;
end;







procedure make_drawings;
Var GraphDriver, GraphMode, ErrorCode, Color,
    Pattern, k, i, j, X1, Y1, X2, Y2         : integer;
    xprev, yprev, mx, my, start_arc, end_arc : real;
    character                                : char;

begin
   GraphDriver:=Detect;
   DetectGraph(GraphDriver,GraphMode);
   InitGraph(GraphDriver, GraphMode,'a:\werk');
   Errorcode:=GraphResult;
   if ErrorCode <> grOK then
   begin
      Writeln('Graphics error  ');
      Writeln(GraphErrorMsg(ErrorCode));
      Writeln('Program aborted ');
      delay(5000);
      Halt(1);
   end;
   clrscr;
   Color:=15;
   SetColor(Color);
   SetFillStyle(Pattern, Color);
   Bar(0,0,619, 399);
   Color:=0;
   SetColor(Color);
   SetFillStyle(Pattern, Color);

   {
   ***********************************************************
   *                                                         *
   *          First the natural parameters will be          *
   *          displayed ( phi and phi_star plot             *
   *          respectively ) and then the Fourier           *
   *          Descriptors ( the amplitude ) .               *
   *                                                         *
   ***********************************************************
   }


   {   here starts the display of the circles and straight lines    }



   xprev:=0;
   yprev:=0;
   for i:=1 to final_number do
   begin
      if final_rec[i].start_angle=final_rec[i].end_angle then
      begin
         X1:=round(xprev);
         Y1:=round(yprev);
         X2:=round(xprev + final_rec[i].radius*cos(final_rec[i].start_angle));
         Y2:=round(yprev + final_rec[i].radius*sin(final_rec[i].start_angle));
         Line(x_screen + X1, y_screen - Y1, x_screen + X2,  y_screen - Y2);
         { delay(2000); }
         xprev:=xprev + final_rec[i].radius*cos(final_rec[i].start_angle);
         yprev:=yprev + final_rec[i].radius*sin(final_rec[i].start_angle);
      end
      else
      begin
         if final_rec[i].orientation=FALSE then
         begin
            start_arc:=(final_rec[i].start_angle - (pi/2))*180/pi;
            end_arc:=(final_rec[i].end_angle - (pi/2))*180/pi;
         end
         else
         begin
            start_arc:=(final_rec[i].start_angle + (pi/2))*180/pi;
            end_arc:=(final_rec[i].end_angle + (pi/2))*180/pi;
         end;


         Draw_Arc(round(start_arc), round(end_arc), xprev, yprev, final_rec[i].radius, i);
      end;
   end;
   repeat;
   until readkey = 'q';
   Closegraph;
end;                 {   of procedure make_drawings  }





procedure analyt;


{
*************************************************************
*                                                           *
*          This procedure inputs a curve interactively      *
*          section by section. A section may be either      *
*          a straight line or a piece of a circle. In       *
*          both cases the starting angle, ending angle,     *
*          starting distance and ending distance are        *
*          interactively inputed piece by piece, after      *
*          which Fourier coefficients are calculated and    *
*          written to a data-file.                          *
*                                                           *
*                                          L.N.'90          *
*                                                           *
*                                                           *
*************************************************************
}





begin
   assign(amdat,'a:\chrdat\fourier\amspec.dat');
   assign(phdat,'a:\chrdat\fourier\phspec.dat');
   assign(g,'a:\chrdat\final.dat');
   reset(g);
   rewrite(amdat);
   rewrite(phdat);
   make_phi_star;
   repeat
      clrscr;
      write('basefrequency f0         : ');
      read(f0);
   until f0>0;
   for k:=1 to max_freq do  { for k=0 division by zero 0, so this limit has to be calculat
   begin
      a:=four_coef_a_star(k);
      b:=four_coef_b_star(k);
      ampl[k]:=sqrt(a*a + b*b);
      if a = 0 then
      begin
         phase[k]:=-pi/2;
         if ((a<0) and (b>0)  or   (a>0) and (b<0)) then phase[k]:=-phase[k];
      end
      else phase[k]:=-arctan(b/a);
   end;
   for k:=1 to max_freq do writeln(amdat,ampl[k]);
   for k:=1 to max_freq do writeln(phdat,phase[k]);
   clrscr;
   make_drawings;
   writeln('program succesfully completed : results are found in the ');
   writeln('A-drive under A:\chrdat\fourier\amspec.dat and under     ');
   writeln('A:\chrdat\fourier\phspec.dat                             ');
   close(g);
   close(amdat);
   close(phdat);
end;



procedure make_draw_file;
var x1, y1, x2, y2, dist, hoek1, hoek2 : real ;   {  x=distance, y=angle }
    first_time                : boolean;

begin
   assign(f,'a:\chrdat\final.dat');
   assign(g,'a:\chrdat\draw.dat');
   reset(f);
   rewrite(g);
   x1:=0;
   y1:=0;
   x2:=0;
   y2:=0;
   first_time:=TRUE;
   while NOT eof(f) do
   begin
      if first_time=TRUE then
      begin
         readln(f,y1);
         readln(f,y2);
         readln(f,x1);
         readln(f,x2);
         writeln(g,x1,'       ',y1);
         writeln(g,x2,'       ',y2);
         readln(f);
         readln(f);
      end
      else
      begin
         readln(f);
         readln(f,y1);
         readln(f);
         readln(f,x1);
         writeln(g,x1,'       ',y1);
         readln(f);
         readln(f);
      end;
      first_time:=FALSE;
   end;
   close(f);
   close(g);
end;


begin           {   MAIN PROGRAM STARTS HERE    }
   make_fills;
   analyt;
   make_draw_file;
end.
```