

## 第七届“认证杯”数学中国

### 数学建模网络挑战赛

#### 承 诺 书

我们仔细阅读了第七届“认证杯”数学中国数学建模网络挑战赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们接受相应处理结果。

我们允许数学中国网站([www.madio.net](http://www.madio.net))公布论文，以供网友之间学习交流，数学中国网站以非商业目的的论文交流不需要提前取得我们的同意。

**我们的参赛队号为：#1031**

**参赛队员（签名）：**

队员 1：沈明

队员 2：王倩

队员 3：王苑

**参赛队教练员（签名）： 无**

**参赛队伍组别：本科组 B 题**

## 第七届“认证杯”数学中国

### 数学建模网络挑战赛

#### 编号专用页

参赛队伍的参赛队号：（请各个参赛队提前填写好）：

#1031

竞赛统一编号（由竞赛组委会送至评委团前编号）：

---

竞赛评阅编号（由竞赛评委团评阅前进行编号）：

# 2014 年第七届“认证杯”数学中国 数学建模网络挑战赛第一阶段论文

题 目 位图矢量化算法研究

关 键 词 矢量重构 径向增量同向点 分段矢量化法 曲线拟合误差

## 摘 要：

在本题中，我们需要将一个形状较为简单的、保存为位图文件的灰度图形矢量化，并表示出矢量图形边界的函数方程。建模过程具体分为以下几步：轮廓提取、跟踪、轮廓特征点确定、轮廓矢量化。

首先，利用索贝尔算子进行边缘检测，提取出边界（轮廓点）；再通过轮廓跟踪将散乱无序的轮廓点建立成有序的点列；在去除有序轮廓点列中冗余点的基础上，采用一种基于以“径向增量同向段”和“径向增量异向段”为基本元素构成位图轮廓边界的提取算法，确定轮廓特征点<sup>[1]</sup>。

在确定轮廓特征点时，相较于离散点求曲率<sup>[2]</sup>等其他特征点提取方法，本文所述的方法大大减少了得到“伪特征点”或丢失真正的特征点使轮廓不能够保形的概率。

根据所找出的轮廓特征点，通过曲线拟合的方式实现图形的矢量化。其中图形的矢量化过程有多种重构方式，诸如多项式（直线、抛物线等），样条曲线和贝塞尔曲线<sup>[4]</sup>等。在轮廓矢量化过程中，我们采用了两种不同的方法：整体矢量化法、分段矢量化法。

整体矢量法将矢量图形边界曲线看作一个整体，运用上述重构方式对特征点进行拟合。此方法拟合出的图形，其边界曲线具有较好的连续性，且较为光滑，但是与原图形边界相比误差较大。

分段矢量法将特征点分段，并对每一段中的特征点分别拟合，最后将分段曲线方程整合为整个曲线的函数方程。它相较于整体矢量化法大大提高了拟合结果的精确度，但可能造成曲线不连续且不闭合，曲线函数方程复杂等情况。

在分段矢量化法下，应用了固定步长法和搜索步长法两种方法对曲线进行分段。在此基础上，又基于曲线拟合误差分析，对每一分段的矢量化重构方式进行了改进：

对每段中的特征点，分别计算这些特征点在各个重构方式下的曲线拟合误差 $R$ ，将使 $R$ 最小的拟合方法作为该段曲线的最优矢量重构法。运用该方法拟合该段曲线的函数方程，大大提高了结果的精确度。

最后，论文对比分析各矢量化方法的优缺点，对模型做出客观评价。

参赛队号： #1031

所选题目： B 题

参赛密码 \_\_\_\_\_  
(由组委会填写)

## Abstract

In this project, our goal is to vectorize a simple-shaped gray-scale figure which is saved as a bitmap file, and give the equation of its contour. The modeling process contains following steps: Contour Extraction, Contour Tracking, Contour Feature-points Extraction and Contour Vectorization.

Firstly, use Sobel operator for edge detection and extract the boundary (contour points). Secondly, rank these scattered and disorderly contour points by contour tracking. Thirdly, after removing redundant points from the set of orderly contour points, we adopt a pick-up algorithm which is based on that and constitute the contour boundary of bitmap as basic elements to extract contour feature-points.

When determining contour feature points, compared with other algorithms of contour feature-points extraction, such as finding the curvature through discrete points, the algorithm in this paper has dramatically reduced the probabilities of getting 'pseudo-feature-points' or losing real feature-points which may result in deformation of the contour.

According to these extracted contour feature-points, the figure vector technique is achieved by means of curve fitting. There are various reconstruction methods of vectorization, such as Polynomial (straight line, parabola etc.), Spline and bezier curve etc. In the process of contour vectorization, we adopt two different methods: entirety vectorization and subsection vectorization.

Entirety vectorization regards the contour of vectorgraph as a whole, and conducts feature-points fitting through reconstruction methods above-mentioned. The boundary curve of a figure fitted in this way has good continuity and is relatively smooth. However, the error will be larger compared to the original contour.

Subsection vectorization conducts feature-points fitting within each part respectively after sectioning these points. The final functional equation of the whole curve is integrated from that of each section. This method improves accuracy of the result enormously compared to the former, while it may cause discontinuity or patency or complex curve functional equation.

Under subsection vectorization, we section the curve through fixed step method and search step method separately. On this basis, the reconstruction method of each section is improved by analyzing error of curve fitting, as follows:

To those feature-points in each section, calculate the error of curve fitting caused by different reconstruction methods respectively. Then we could regard the method as an optimal one which makes the error smallest. The experiment shows that the accuracy of results is improved a lot through this method.

Last but not least, this paper evaluates models objectively by comparing the merits and faults of various methods of vectorization.

## 一、问题重述

### § 1.1 问题背景

图形（图像）在计算机中主要有两种存储和表示方法：矢量图和位图。前者是使用点、直线或多边形等基于数学方程的几何对象来描述图形，它经过任意放缩后不会有任何改变；而后者则使用像素来描述图像，一旦将其放大后就会产生较为明显的模糊，线条也会出现锯齿边缘等现象。

矢量图从本质上只是使用曲线方程对图形进行的精确描述，在以像素为基本显示单元的显示器或打印机上是无法直接表现的，需要将它转换成以像素点阵来表示的信息，再加以显示或打印。这个过程称之为栅格化，目前已完全掌握这项技术。但是，栅格化的逆过程相对而言比较困难。

### § 1.2 需要解决的问题

栅格化的逆过程：通过建立数学模型，将一个形状较为简单的、保存为位图文件的灰度图形矢量化：

1. 提取栅格图像边界（轮廓），得到无序轮廓点，并跟踪；
2. 根据跟踪所得有序轮廓点提取矢量图形边界特征点；
3. 根据矢量图形边界特征点拟合得到矢量图形边界的函数方程。

## 二、模型假设

1. 位图图像为灰度图像；
2. 矢量图形为简单图形，图形没有悬挂点和悬挂边且不包含岛；
3. 矢量图形边界由基本曲线组合而成；
4. 矢量图形边界封闭且不自相交。

## 三、符号定义和说明

$\{p_n\}$	有序轮廓点列，其中 $p_i$ 表示该点列中的第 $i$ 个点
$SDRIC$	径向增量同向点
$SDRID$	径向增量异向点
$k$	相邻两点连线的斜率
$k_i$	第 $i$ 个点与第 $i+1$ 个点连线的斜率
$p.x$	点 $p$ 横坐标
$p.y$	点 $p$ 纵坐标

$D_i$	相邻直线段的长度
$\Delta$	两点间的间隔距离
$\coprod^e$	等间隔点集
$T$	特征点集
$B$	固定步长
$R$	曲线拟合误差

#### 四、问题分析与求解

位图矢量化可以分为轮廓提取、跟踪、轮廓特征点确定、轮廓矢量化。

##### § 4.1 轮廓的提取

**轮廓提取：**将位图转换为灰度图，利用索贝尔算子（Sobel operator）对灰度图像进行边缘检测，提取出边界（轮廓）；



图1 灰度图

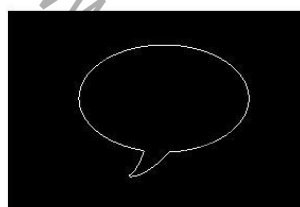


图2 边缘检测后的轮廓图

##### § 4.2 轮廓的跟踪

###### ➤ 4.2.1 轮廓跟踪的分析

为将轮廓提取后所得到的一些散乱的无序轮廓点建立成有序的点列，需要进行轮廓跟踪。

###### ➤ 4.2.2 轮廓跟踪的求解

利用张伟刚等人所著论文<sup>[3]</sup>中的如下算法进行追踪：  
（具体算法步骤见表1）

表 1 轮廓跟踪算法

算法步骤	流程图
<p>T1. 确定起点 S;</p> <p>T2. 初始化: 当前点 <math>C \leftarrow S</math>, 上一点 <math>L \leftarrow S</math>, <math>i \leftarrow 1</math>;</p> <p>T3. 求跟踪目标: 求 C 邻域内非 L 的邻点 N 为跟踪目标;</p> <p>T4. 比较 S、C: 若 <math>S \neq C</math>, 则将 L 压入堆栈, 且 <math>L \leftarrow C</math>, <math>C \leftarrow N</math>, <math>i \leftarrow i+1</math>, 并返回步骤 T3;</p> <p>T5. 对 j 循环: 置 <math>j \leftarrow 1, 2, \dots, i</math>, L 弹出堆栈, <math>TP(j) \leftarrow L</math>, 若堆栈为空, 算法告终; 于是, 得到有序轮廓点集。</p>	<pre> graph TD     Start([Start]) --&gt; FindS[寻找未被访问的轮廓点S]     FindS --&gt; Init[C ← S, L ← S, i = 1]     Init --&gt; FindN[跟踪目标 求C邻域内非C邻点N]     FindN --&gt; SC{S = C}     SC -- N --&gt; Update[L ← C C ← N i ← i + 1]     Update --&gt; FindN     SC -- Y --&gt; InN{i &lt; n}     InN -- Y --&gt; LoopJ[对j循环 置j ← 1, 2, ... i L弹出堆栈]     LoopJ --&gt; InN     InN -- N --&gt; Output[输出有序轮廓点集]   </pre>

### § 4.3 轮廓特征点的确定

#### ➤ 4.3.1 轮廓特征点确定的分析

由于矢量图是使用点、直线或光滑曲线等基于数学方程的几何对象来描述图形的, 即矢量图形的边界是由几段直线或光滑曲线相接而成。在矢量图形边界的有序轮廓点列  $\{p_n\}$  中, 若某点在某一基本曲线上且既不是该基本曲线的端点也不能表示曲线的突变特征, 则称该点为冗余点; 反之, 则称其为特征点。

显然, 特征点是表征图形轮廓突变特征的点, 体现了图形边缘轮廓的几何形状。因此, 我们首先需要确立图形的轮廓特征点, 再对特征点集进行矢量化, 得到重构的矢量图形。

严素蓉等所著论文<sup>[1]</sup>中, 采用一种基于以“径向增量同向段”和“径向增量异向段”为基本元素构成位图轮廓边界的轮廓特征点提取算法。此算法相较于离散点求曲率等其他特征点提取方法, 大大减少了得到“伪特征点”或丢失真正的特征点使轮廓不能够保形的概率。因此, 本文采用该算法进行特征点的提取。

首先对“径向增量同向段”与“径向增量异向段”进行定义<sup>[1]</sup>:

给定一组离散序列  $\{p_n\}$ , ( $n > 4$ ), 如果满足  $p_i$  在以  $p_{i+1}$  为圆心的某圆周上,  $p_{i+3}$  在以  $p_{i+2}$  为圆心的某圆周上, 且  $p_i, p_{i+3}$  相对各自所在圆的圆心增量在 X 或 Y 方向同号,

则称  $p_{i+1}, p_{i+2}$  为“径向增量同向点”（**虚**）， $p_i, p_{i+1}, p_{i+2}, p_{i+3}$  所构成的曲线段为径向增量同向段（如图一虚线线段）。否则，则称  $p_{i+1}, p_{i+2}$  为“径向增量异向点”（**实**）， $p_i, p_{i+1}, p_{i+2}, p_{i+3}$  所构成的曲线段为径向增量异向段（如图 3 实线线段）。

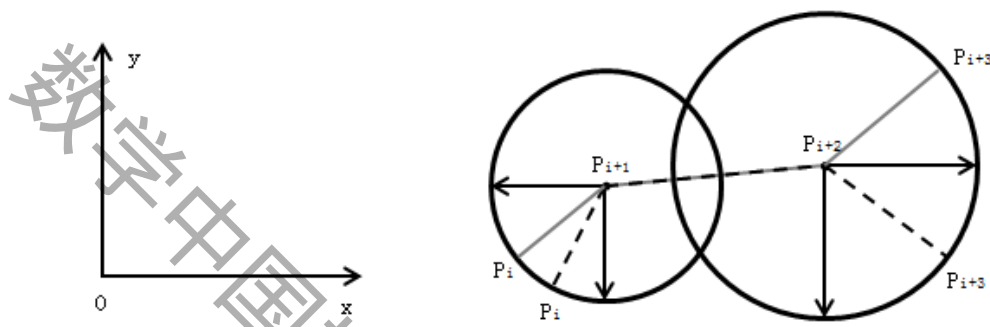


图 3 “径向增量同向段” 和 “径向增量异向段”

#### ➤ 4.3.2 轮廓特征点确定的求解

##### 1、冗余点的清除（具体算法见下文表 2）

对于初始的轮廓点而言：

取四个特征点  $p_1, p_2, p_3, p_4$ ，其中  $k_i$  为第  $i$  个点与第  $i+1$  个点间连线的斜率（对于竖

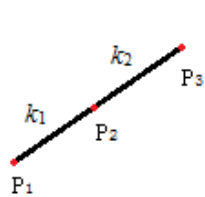
直线，其斜率  $k = +\infty$ ）， $i = 1$ 。在严素蓉等所著论文<sup>[1]</sup>中，考虑了情形一和情形二以去除冗余点，但是没有考虑到情形三，即由多组径向增量异向段顺序组成的规则的曲折线段，其点列可能成梯度排列的情况。对比图 5（b）和图 5（c）可发现，考虑情形三能够更有效地去除冗余点。

**情形一：**斜率相等（ $k_1 = k_2$ ），即  $p_1, p_2, p_3$  在同一直线上，可认为  $p_2$  是冗余点（如图 4（a））；

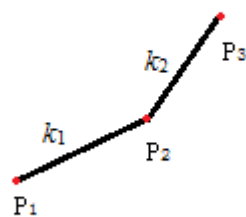
**情形二：**斜率同号（ $k_1 \times k_2 > 0$ ），即  $p_1, p_2, p_3$  在同一曲线上，可认为  $p_2$  是冗余点（如图 4（b））；

**情形三：** $k_1 = k_3$  且  $k_2 = 0$  或  $+\infty$  时，它们实质代表一条直线，所以可认为  $p_2, p_3$  为冗余点（如图 5（a））。



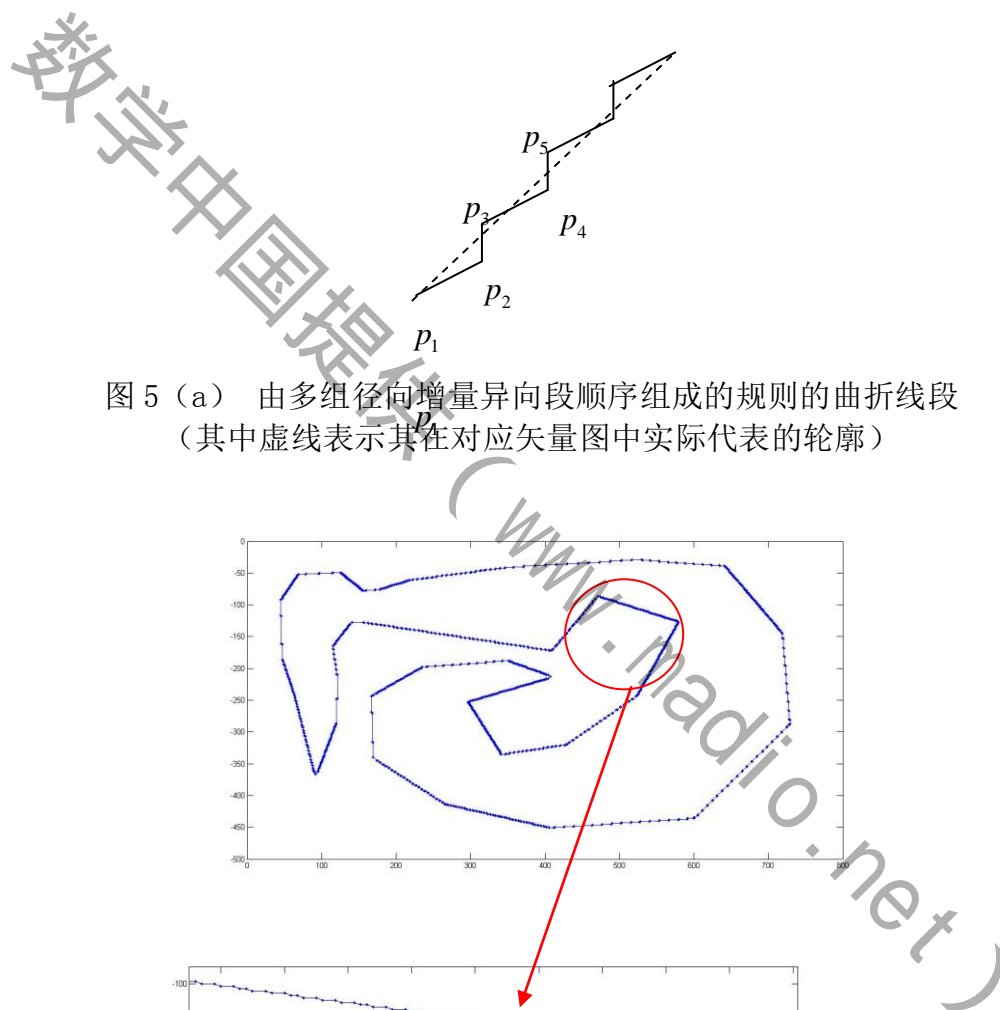
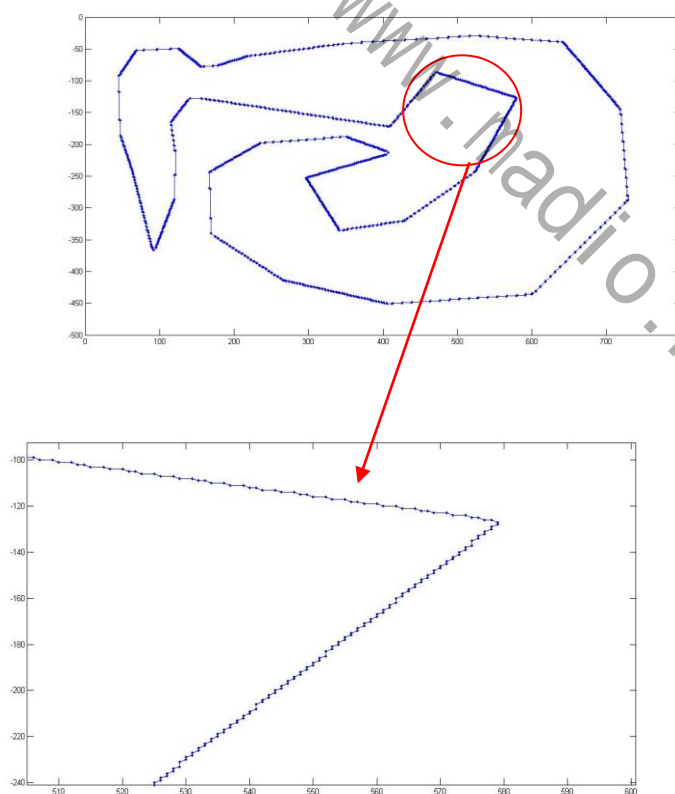


(a)



(b)

图4 冗余点的两种情形

图5 (a) 由多组径向增量异向段顺序组成的规则的曲折线段  
(其中虚线表示其在对应矢量图中实际代表的轮廓)图5 (b) 仅考虑  $k_1 = k_2$  和  $k_1 \times k_2 > 0$  两种情况下去除冗余点后的效果图

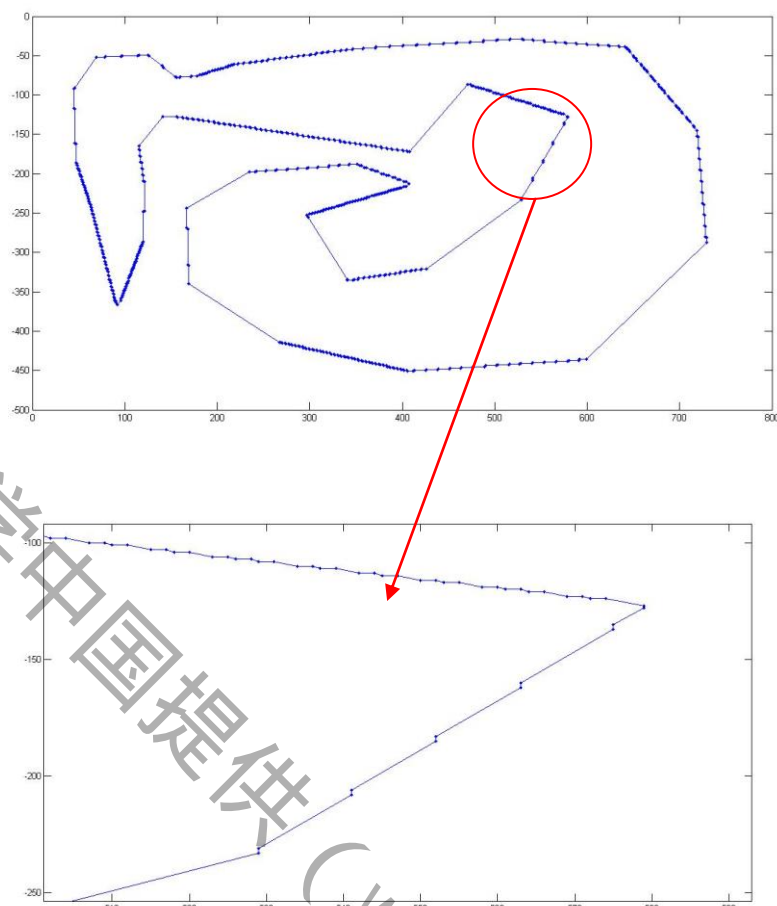


图 5 (c) 考虑多组径向增量异向段顺序组成的规则的曲折线段去除冗余点后的效果图

## 2、提取 SDRIC 点和长直线段端点（具体算法<sup>[1]</sup>见下文表 3）

分析图形的边缘轮廓构成，可把它细分为由径向增量同向段和由一组或多组径向增量异向段顺序组成的曲折线段构成。径向增量同向段主要代表了轮廓的突变特征（如图 6 (a)）；有些径向增量异向段中，包含了由径向增量异向点组成的较长线段，这些线段也表示了轮廓的主要特征（如图 6 (b)）；因此特征点提取算法可以认为是从有序轮廓点中提取径向增量同向点和径向增量异向段中的长直线段的端点。

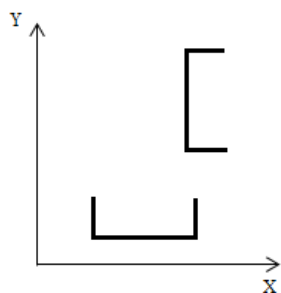


图 6 (a) 径向增量同向段

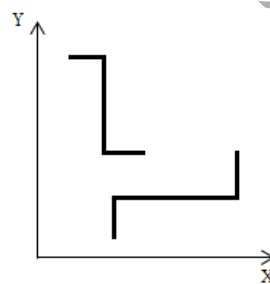


图 6 (b) 由径向增量异向点组成的较长线段

表 2 冗余点清除算法

算法步骤	流程图
<p>对于有序轮廓点列</p> $\{p_i\} = p_0 \cdots p_{i-1}, p_i, p_{i+1} \cdots p_n, \text{ 令}$ $sp = p_0, mp = p_1, tp = p_2, fp = p_3$ <p>, 循环变量 <math>i = 0</math> 重复执行步骤 1, 2, 3, 4 直到 <math>i &gt; n - 4</math> 时终止。</p> <p>S1: 求相邻线段的斜率:</p> $k_1 = (mp.y - sp.y) / (mp.x - sp.x)$ $k_2 = (tp.y - mp.y) / (tp.x - mp.x)$ $k_3 = (fp.y - tp.y) / (fp.x - tp.x)$ <p>S2: 合并斜率相等的线段: 如果 <math>k_1 = k_2</math>, 把 <math>mp</math> 作为冗余点从轮廓点去除, 跳到 S5;</p> <p>S3: 合并斜率同号的线段: 如果 <math>k_1 \times k_2 &gt; 0</math> 把 <math>mp</math> 作为冗余点从轮廓点去除, 跳到 S5;</p> <p>S4: 合并由多组径向增量异向段顺序组成的规则的曲折线段:</p> <p>如果 <math>k_1 = k_3</math> 且 <math>k_2 = 0</math> 或 <math>+\infty</math>,</p> <p>则把 <math>mp, tp</math> 作为冗余点从轮廓点中去除, 跳到 S5;</p> <p>S5: 否则, 线段后移: <math>i = i + 1</math></p> $sp = p_i, mp = p_{i+1},$ $tp = p_{i+2}, fp = p_{i+3}$	<pre> graph TD     Start([Start]) --&gt; Init[sp = p0, mp = p1, tp = p2, fp = p3]     Init --&gt; I0[i = 0]     I0 --&gt; Cond1{i &lt; n - 3}     Cond1 -- N --&gt; Output[输出无冗余点的轮廓点]     Cond1 -- Y --&gt; CalcK[ k1 = (mp.y - sp.y) / (mp.x - sp.x) k2 = (tp.y - mp.y) / (tp.x - mp.x) k3 = (fp.y - tp.y) / (fp.x - tp.x)]     CalcK --&gt; Cond2{k1 = k2}     Cond2 -- Y --&gt; RemoveMP[去除 mp]     Cond2 -- N --&gt; Cond3{k1 = k3 且 k2 = 0 或 +∞}     Cond3 -- Y --&gt; RemoveMPTP[去除 mp, tp]     Cond3 -- N --&gt; Cond4{k1 × k2 &gt; 0}     Cond4 -- Y --&gt; RemoveMP     Cond4 -- N --&gt; Update[i = i + 1, sp = pi, mp = pi+1, tp = pi+2, fp = pi+3]     RemoveMP --&gt; Update     RemoveMPTP --&gt; Update     Update --&gt; Cond1   </pre>

表 3 提取 SDRIC 点和长直线段端点算法

算法步骤	流程图
<p>对于有序轮廓点列</p> $\{p_i\} = p_0 \cdots p_{i-1}, p_i, p_{i+1} \cdots p_n, \text{ 令}$ $sp = p_0, \quad mp = p_1, \quad flag = 0,$ $i = 2。$ <p>用 <math>\sigma = \sum_{i=0}^n \ p_i - p_{i+1}\  / n + h</math> (<math>h</math> 微调 <math>\sigma</math> 的整数, 范围: <math>0 \sim 5</math>) 来判断两点的距离 <math>D_0 = \ p_i - p_{i+1}\ </math> 是否满足长直线段条件。</p> <p>S1: 条件变量计算: <math>tp = p_i,</math></p> $fp = p_{i+1};$ <p>相邻两点间直线段的长度:</p> $D_0 = \ sp - mp\ ,$ $\delta = (sp.x - mp.x) \times (fp.x - tp.x) > 0$ $\  (sp.y - mp.y) \times (fp.y - tp.y) > 0$ <p>判断条件变量是否满足下列约束条件;</p> <p>S2: 提取 SDRIC 点: 若 <math>\delta</math> 为真, 则 <math>mp, tp</math> 为特征点, <math>flag = 0</math>;</p> <p>条件变量代换: <math>sp = tp,</math></p> $mp = fp, \quad i = i + 2;$ <p>S3: 提取长直线段:</p> <p>若 <math>(D_0 &gt; \sigma)</math> 为真, 则 <math>sp, mp</math> 为特征点, <math>flag = 0</math>, 跳至 S4;</p> <p>S4: 否则, 进行条件变量代换:</p> $sp = mp, \quad mp = tp, \quad i = i + 1。$	<pre> graph TD     Start([Start]) --&gt; Init[sp = p0, mp = p1, flag = 0, i = 2]     Init --&gt; Cond1{i+1 &gt; n}     Cond1 -- Y --&gt; Output[输出特征点]     Cond1 -- N --&gt; Calc[tp = pi, fp = pi+1, D0 =   sp - mp   delta = (sp.x - mp.x) * (fp.x - tp.x) &gt; 0    (sp.y - mp.y) * (fp.y - tp.y) &gt; 0]     Calc --&gt; Cond2{delta = 1}     Cond2 -- Y --&gt; Set1[mp, tp 为特征点 flag = 0]     Set1 --&gt; Update1[sp = tp, mp = fp, i = i + 2]     Update1 --&gt; Cond1     Cond2 -- N --&gt; Cond3{D0 &gt; sigma}     Cond3 -- Y --&gt; Set2[sp, mp 为特征点 flag = 0]     Set2 --&gt; Update2[sp = mp, mp = tp, i = i + 1]     Update2 --&gt; Cond1     Cond3 -- N --&gt; Update2   </pre>

当  $h$  取 5 时, 特征点集的效果如图 7 所示。

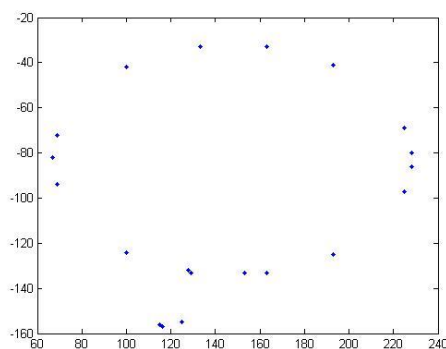
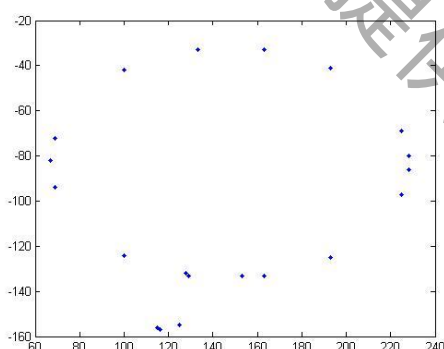
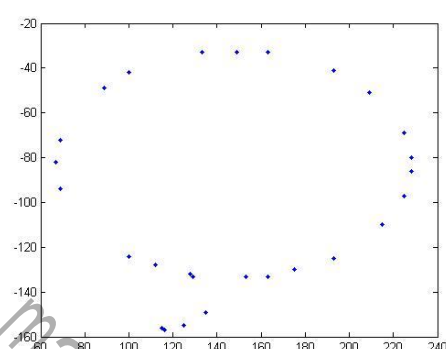


图7 h取5时特征点集的效果图

### 3、插值

上述所提取的特征点并不是完全充分的，图形的特征点可能过于稀少，而导致重构的轮廓失真（如图8（a）），因此有必要在特征点列中进行插值<sup>[1]</sup>，也即在原始轮廓点列中，保留一些等 $\Delta$ 间隔点，这些点与上述的特征点进行有条件的归并。因为是原始轮廓点，所以得到的最终轮廓特征点列可以很好地保留轮廓原始信息，减小失真。

图8（a）插值前（ $h=5$ ）图8（b）插值后（ $h=5, \varepsilon=10$ ）

具体算法<sup>[1]</sup>如下：

设对原始轮廓点数 $n$ 大于 $\Delta$ 的轮廓序列点，把每间隔 $\Delta$ 的点提取出来放入等间隔点集 $\coprod e$ 中。其中 $\Delta = \sigma \pm \varepsilon$ ， $\sigma$ 是该轮廓中提取长直线段时的下限值， $\varepsilon$ 是微调 $\Delta$ 的正数值，范围：0~10， $\coprod e = \{p_{\Delta}, p_{2\Delta}, \dots, p_{m\Delta}\}$ （ $m\Delta < n$ ）

对于等间隔点集 $\coprod e$ 中的每一元素，如果它到位于其前面和后面的特征点的距离都大于 $\frac{\Delta}{2}$ ，则保留并加入到最终轮廓特征点集 $T$ 中。

$$T = (\coprod_{i=0}^n c_i) \cup (\coprod_{j=0}^m e_j)$$

其中， $c_i$  是下标为  $i$  ( $0 \leq i \leq n$ ) 的特征点， $e_j$  是下标为  $j$  ( $0 \leq j \leq m$ ) 并满足  $(\|e_j - c_i\| > \frac{\Delta}{2})$  且  $(\|e_j - c_{i+1}\| > \frac{\Delta}{2})$  的等间隔点。如图中的 (b) 不但包括了描述轮廓形状的关键点，达到轮廓形状的保形，而且较均匀分步的轮廓点，使下一步采用何种矢量形体来拟合变得简单。

## § 4.4 轮廓的矢量化

### ➤ 4.4.1 轮廓矢量化的分析

根据前文所找出的轮廓特征点 ( $h=2, \varepsilon=3$  时的情形)，通过曲线拟合的方式实现图形的矢量化。其中图形的矢量化过程有多种方法，诸如多项式（直线、抛物线等），样条曲线和贝塞尔曲线<sup>[4]</sup>等。

### ➤ 4.4.2 轮廓矢量化的求解

#### 1、整体矢量化法

将矢量图形边界曲线看做一个整体，运用上述重构方法对特征点进行拟合。本文以贝塞尔曲线重构法为例对曲线的所有特征点进行拟合，得到了一条光滑的曲线（图 9），曲线方程为：

$$B(t) = \sum_{i=0}^n \binom{n}{i} p_i (1-t)^{n-i} t^i, \quad t \in [0,1] \quad (1)$$

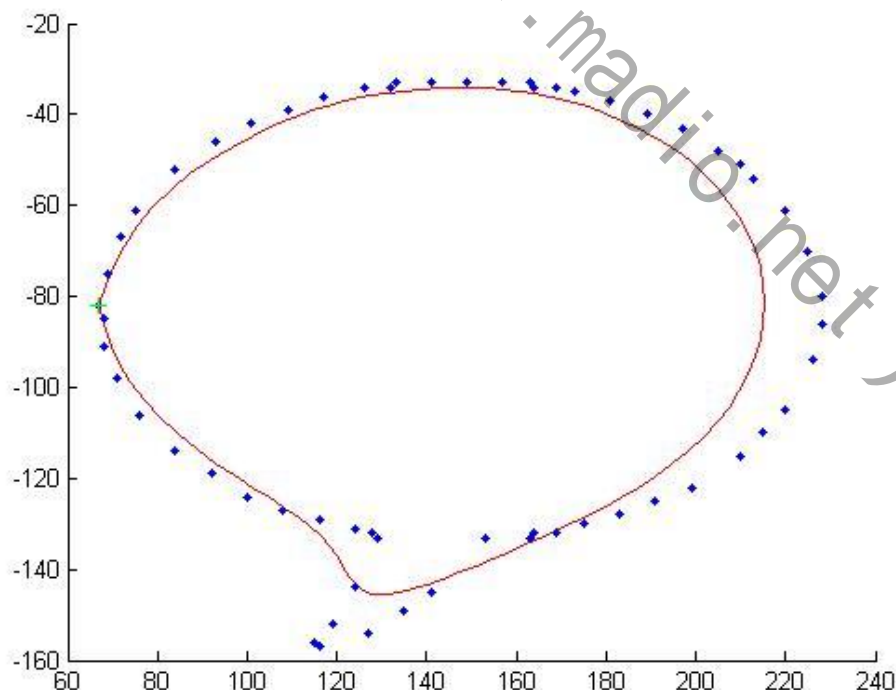


图 9 整体矢量化法下贝塞尔曲线拟合

## 2、分段矢量化法

尽管采用整体矢量化法拟合出的图形边界曲线具有较好的连续性，且比较光滑，但是与原图形边界相比误差较大。为减小曲线拟合的误差，我们考虑采用将曲线进行分段拟合的方式，将每一段中的特征点分别拟合，到达矢量化的目的。

### (1) 固定步长法

将特征点按固定的步长分段后，把每段中的特征点按贝塞尔曲线法、多项式法、样条曲线法等矢量化重构方法分别进行矢量化。（具体算法见表4）

表4 固定步长法算法

算法步骤	流程图
<p>S1: 设特征点总数为<math>n</math>，选取固定的步长<math>B(\geq 3)</math>，将特征点分为：</p> $\begin{cases} n/B & n/B \text{ 为整数} \\ [n/B]+1 & n/B \text{ 不为整数} \end{cases} \text{ 段；}$ <p>S2: 对每段中的特征点采用选定的矢量重构方法分别进行拟合，得到分段曲线函数方程；</p> <p>S3: 整合分段曲线函数方程得到矢量图形边界函数方程。</p>	<pre> graph TD     Start([Start]) --&gt; Init[i=1, B=0]     Init --&gt; Decision{i+B &lt; n}     Decision -- Y --&gt; Box1["x = (p_i.x, ..., p_{i+B}.x) y = (p_i.y, ..., p_{i+B}.y)"]     Box1 --&gt; Box2["利用多项式或贝塞尔曲线 拟合x、y, 得到该段函数方程"]     Box2 --&gt; Inc[i = i + B]     Inc --&gt; Decision     Decision -- N --&gt; Box3["拟合尾部 x = (p_i.x, ..., p_n.x) y = (p_i.y, ..., p_n.y)"]     Box3 --&gt; Box4["拟合x、y 得到该段曲线方程"]     Box4 --&gt; Output[输出曲线方程]   </pre>

下列图形是 $B=3$ 时，分别采用一次多项式拟合（见下页图10（a）），二次多项式拟合（图10（b）），贝塞尔曲线拟合（图10（c））的矢量重构方法所得到的矢量图。

从图中可以看出，与整体矢量化方法相比，分段拟合具有较好的精确性，误差较小。但由此拟合得到的曲线是不连续且不闭合的，需要在相邻曲线端点处添加竖直线段以使图形闭合（如图11）。

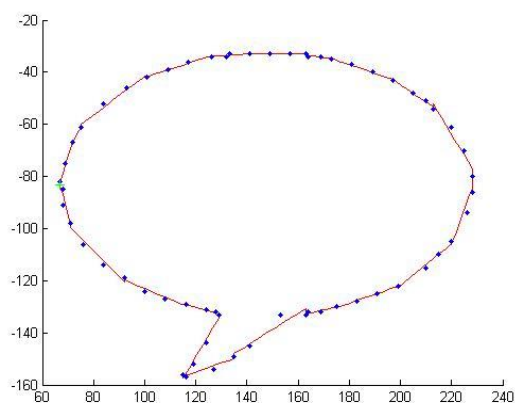


图 10 (a) 固定步长法下一次多项式拟合

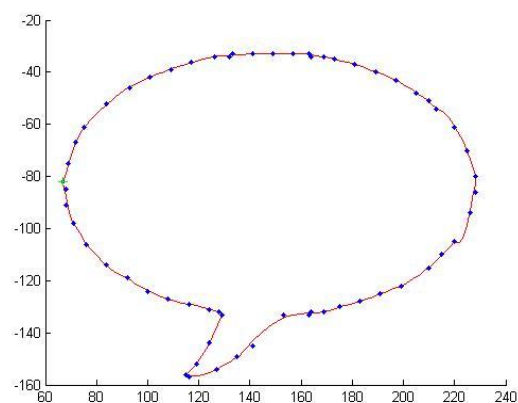


图 10 (b) 固定步长法下二次多项式拟合

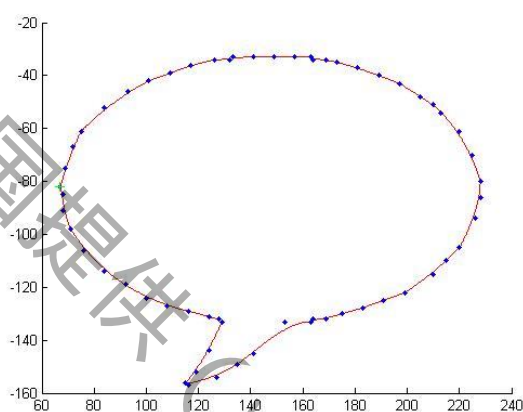


图 10 (c) 固定步长法下贝塞尔曲线拟合

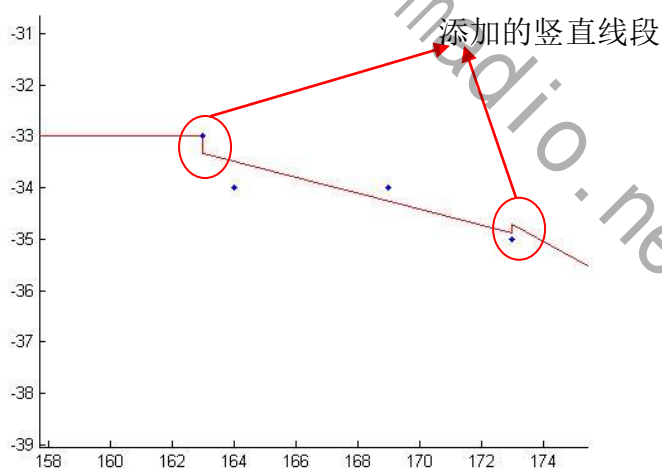


图 11 在相邻曲线端点处添加竖直线段后的效果图

在三种拟合方法中，贝塞尔曲线拟合效果最佳。

在  $B=3$  时，通过贝塞尔曲线法拟合函数（各段曲线拟合后的参数见表 5），得到的第  $i$  段曲线方程为：

$$B_i(t) = p_{i_0}(1-t)^3 + 3p_{i_1}t(1-t)^2 + 3p_{i_2}t^2(1-t) + p_{i_3}t^3, \quad t \in [0,1] \quad (2)$$



表 5：贝塞尔曲线重构法中各段曲线拟合后的参数

	P0(x, y)		P1(x, y)		P2(x, y)		P3(x, y)	
1	67	-82	68	-85	68	-91	71	-98
2	71	-98	76	-106	84	-114	92	-119
3	92	-119	100	-124	108	-127	116	-129
4	116	-129	124	-131	128	-132	129	-133
5	129	-133	124	-144	119	-152	115	-156
6	115	-156	116	-157	127	-154	135	-149
7	135	-149	141	-145	153	-133	163	-133
8	163	-133	164	-132	169	-132	175	-130
9	175	-130	183	-128	191	-125	199	-122
10	199	-122	210	-115	215	-110	220	-105
11	220	-105	226	-94	228	-86	228	-80
12	228	-80	225	-70	220	-61	213	-54
13	213	-54	210	-51	205	-48	197	-43
14	197	-43	189	-40	181	-37	173	-35
15	173	-35	169	-34	164	-34	163	-33
16	163	-33	157	-33	149	-33	141	-33
17	141	-33	133	-33	132	-34	126	-34
18	126	-34	117	-36	109	-39	101	-42
19	101	-42	93	-46	84	-52	75	-61
20	75	-61	72	-67	69	-75	67	-82

## (2) 基于误差分析改进的固定步长法

尽管与整体矢量化方法相比，固定步长法在一定程度上提高了曲线拟合的精度，但是仍然存在很大的误差。为此，需要改进固定步长算法，实现在各段特征点中分别选取最优的矢量重构法以提高精度。本文引入误差分析的方法来改进固定步长算法。

首先给出曲线拟合误差的定义：

设  $n$  个特征点  $\{p_i\}, i=1 \cdots n$  利用某种重构方法拟合出的曲线为  $f(x)$ ，则称 (3) 式为这  $n$  个特征点在该重构方法下曲线拟合误差。

$$R = \sum_{i=1}^n \frac{(f(x_i) - p_i \cdot y)^2}{n} \quad (3)$$

具体算法如下：

S1：设特征点总数为  $n$ ，选取固定的步长  $B(\geq 3)$ ，将特征点分为

$$\begin{cases} n/B & n/B \text{ 为整数} \\ [n/B]+1 & n/B \text{ 不为整数} \end{cases} \text{ 段；}$$

S2：对每段中的特征点，分别计算这些特征点在各个重构方法下的曲线拟合误差  $R$ ，将误差  $R$  最小的拟合方法作为本段曲线的最优矢量重构法，运用该方法拟合本分段曲线的

函数方程；

S3：整合分段曲线函数方程得到矢量图形边界函数方程。

下列图形是使用固定步长法分别通过一、二次多项式拟合（图 12（a）、图 12（b））和基于误差分析拟合（图 12（c））的效果对比：

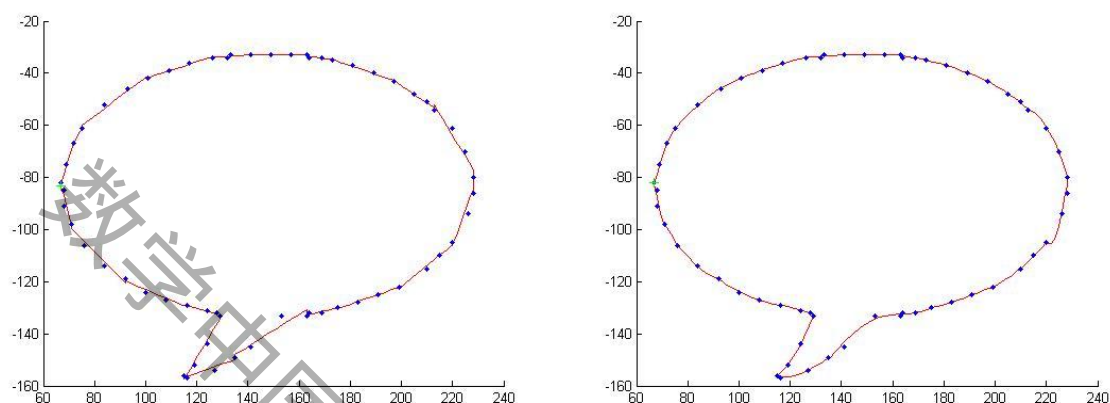


图 12(a) 固定步长法改进前一次多项式拟合 图 12(b) 固定步长法改进前二次多项式拟合

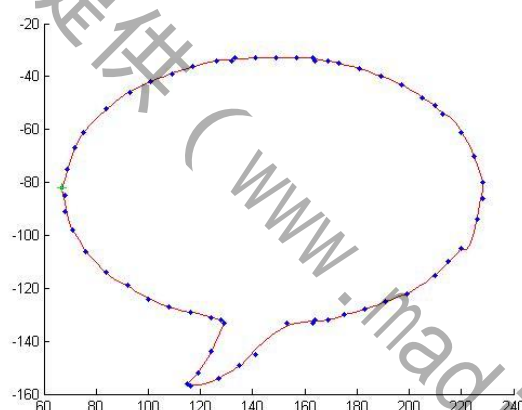


图 12(c) 基于误差分析拟合（以采用一次、二次多项式拟合为例）

在  $B=3$  时，基于误差分析的多项式拟合函数方程（各段曲线拟合后的参数见表 6），第  $i$  段曲线方程为：

$$y_i = ax_i^2 + bx_i + c. \quad (4)$$

表 6：基于误差分析的多项式拟合法中各段曲线拟合后的参数

	$x_0$	$x_1$	$a$	$b$	$c$
1	67	71	0.66666667	-96	3357.333
2	71	92	0.03147578	-6.118078	177.5403
3	92	116	0.01171875	-2.85	43.9625
4	116	129	-0.0103572	2.2500362	-250.651
5	129	115	0.06287559	-13.6965	587.549

	$x_0$	$x_1$	$a$	$b$	$c$
6	115	135	0.0223396	-5.211059	147.3887
7	135	163	-0.0241292	7.815659	-765.369
8	163	175	0.01451021	-4.69173	246.6403
9	175	199	0.00390625	-1.123438	-53.0773
10	199	220	0.01596371	-5.873291	414.578
11	220	228	0.45833333	-202.5833	22280
12	228	213	-0.0993939	42.137576	-4520.03
13	213	197	-0.0103044	3.5523674	-342.956
14	197	173	-0.0039062	1.1078125	-109.691
15	173	163	-0.0051515	1.5754545	-153.31
16	163	141	-3.69E-17	1.11E-14	-33
17	141	126	-0.0015957	0.4969927	-71.3276
18	126	101	-0.0048053	1.4154818	-136.008
19	101	75	-0.0144738	3.2728338	-224.984
20	75	67	-0.1318898	21.341864	-919.793

(接上页表 6)

### (3) 搜索步长法（具体算法见表 7）

尽管固定步长的分段曲线拟合具有较高的精度，但是在给定曲线方程时，如果特征点数目较多，会导致方程个数过多，结果较为复杂。在此引入搜索步长的概念，自动寻找最优步长，以减少曲线方程的复杂度。

图 13 是采用搜索步长法，通过多项式拟合得到的结果：

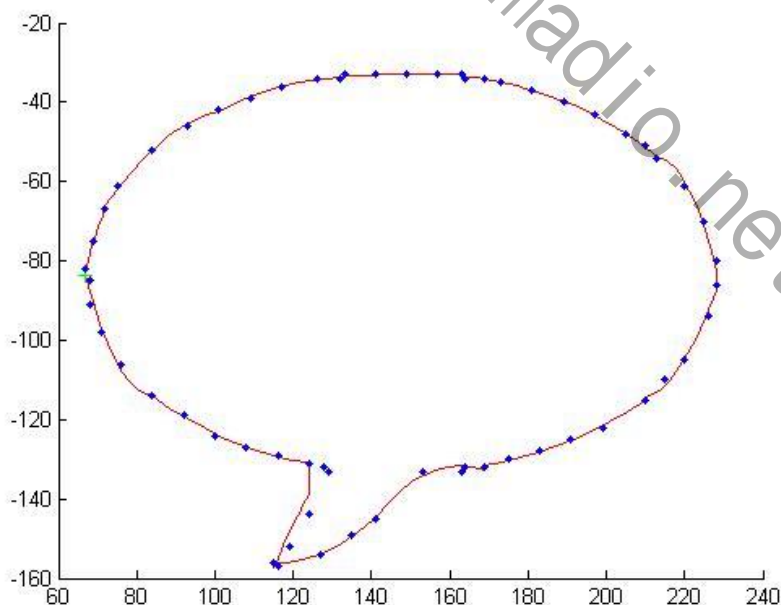


图 13 采用搜索步长法通过多项式拟合的效果图

表 7 搜索步长法算法

算法步骤	流程图
<p>S1: 设特征点总数为 <math>n</math>，选取固定的步长 <math>B_0</math>；</p> <p>S2: 基于误差分析得到本段曲线初始特征点集的最优重构方法，并计算这些特征点在该重构方法下的曲线拟合误差 <math>R_0</math>；</p> <p>S3: 每次使步长加 1，即每次增加一个特征点到本段曲线特征点集 <math>T_i</math> 中，计算这些特征点在本最优重构方法下的误差 <math>R</math>，直到 <math>R &gt; R_0</math> 时不再增加步长；</p> <p>S4: 利用本段曲线的最优重构方法，对该特征点集 <math>T_i</math> 进行拟合得到第 <math>i</math> 段曲线的函数方程；如果所有的特征点均被使用则跳至 S5，否则返回 S2；</p> <p>S5: 整合分段曲线函数方程得到矢量图形边界函数方程。</p>	<pre> graph TD     Start([Start]) --&gt; Init[B = B0, i = 1]     Init --&gt; Cond1{i + B &lt; n}     Cond1 -- N --&gt; Out1[输出整个曲线的方程]     Cond1 -- Y --&gt; Calc[x = pi * x ... pi_{i+B} * x y = pi * y ... pi_{i+B} * y]     Calc --&gt; Fit1[一次多项式拟合曲线，计算误差 R01 二次多项式拟合曲线，计算误差 R02]     Fit1 --&gt; Cond2{R01 &lt; R02}     Cond2 -- Y --&gt; Fit2[本段采用一次多项式拟合 R0 = R01]     Cond2 -- N --&gt; Fit3[本段采用二次多项式拟合 R0 = R02]     Fit2 --&gt; Calc2[B = B + 1 tx = (pi * x, ..., pi_{i+B} * x) ty = (pi * y, ..., pi_{i+B} * y) 拟合 x、y，计算误差 R]     Fit3 --&gt; Calc2     Calc2 --&gt; Cond3{R &lt; R0}     Cond3 -- Y --&gt; Calc3[x = tx y = ty B = B + 1 tx = (pi * x, ..., pi_{i+B} * x) ty = (pi * y, ..., pi_{i+B} * y) 拟合 tx、ty，计算误差 R]     Cond3 -- N --&gt; Out2[拟合 x, y 计算得到本段方程]     Out2 --&gt; Out1   </pre>

在  $B=3$  时，基于搜索步长的多项式拟合函数方程（各段曲线拟合后的参数见表 8），第  $i$  段曲线方程为：

$$y_i = ax_i^2 + bx_i + c \quad (5)$$

表 8：基于搜索步长的多项式拟合法中各段曲线拟合后的参数

	$x_0$	$x_1$	$a$	$b$	$c$
1	67	84	0.1081	-18.0805	642.3149
2	84	124	0.007533	-1.98839	-0.09554
3	124	115	-0.03868	11.29647	-944.978
4	115	141	0.015876	-3.61323	49.03665
5	141	169	-0.02858	9.283601	-885.445
6	169	210	0.005451	-1.66516	-5.93264
7	210	228	0.066241	-27.503	2739.961
8	228	213	-0.13737	58.69744	-6324.47
9	213	181	-0.00865	2.8894	-276.899
10	181	163	-0.00562	1.734086	-166.599
11	163	132	-0.00154	0.469774	-68.6839
12	132	101	-0.00738	1.990456	-167.869
13	101	72	-0.0206	4.386711	-275.209
14	72	67	-0.16667	26.16667	-1087

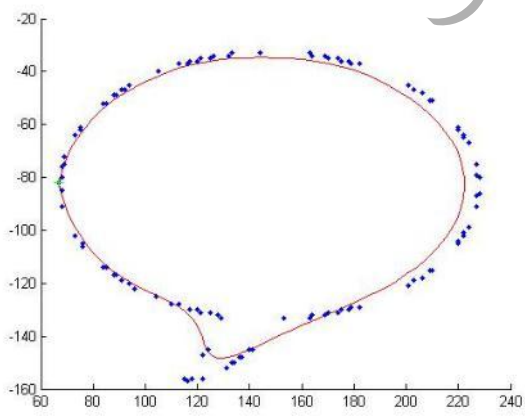
由表 8 可以看出，基于搜索步长的多项式拟合法较固定步长的多项式拟合法而言，大大减少了曲线的分段数目，简化了方程。

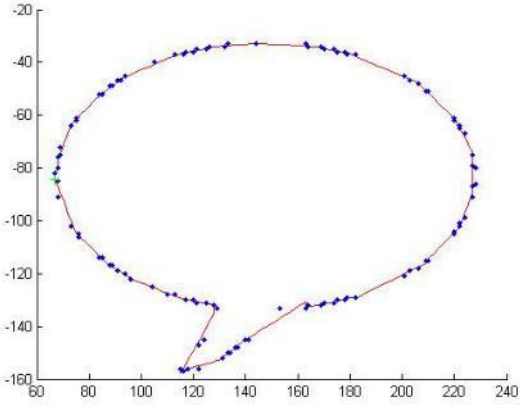
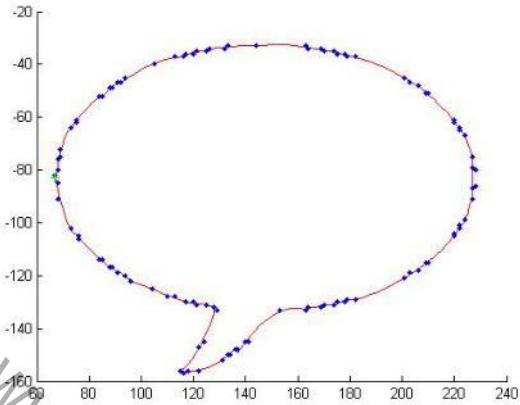
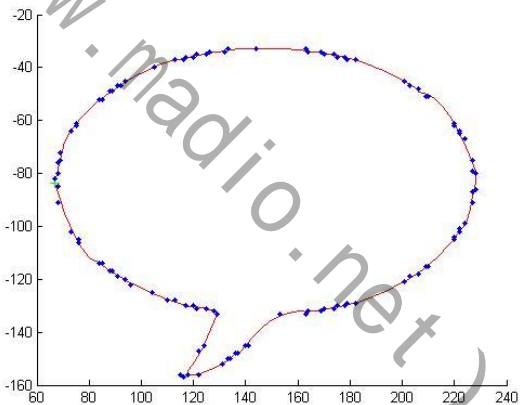
## 五、模型评价

在提取特征点时，我们参照严素蓉等人的论文<sup>[1]</sup>，采用了基于以“径向增量同向段”和“径向增量异向段”为基本元素构成位图轮廓边界的轮廓特征点提取算法。此算法相较于离散点求曲率等其他特征点提取方法，大大减少了得到“伪特征点”或丢失真正的特征点使轮廓不能够保形的概率。

而在轮廓矢量化过程中，我们分别采用：1. 整体法，2. 分段法：固定步长法、搜索步长法进行矢量化，其中还通过误差分析的方式改进了固定步长法，以下是上述几种矢量化方法的优缺点比较：

表 9 几种矢量化方法的优缺点比较

方法名称		优点	缺点	图示
整体矢量化法		1. 能得到连续光滑的曲线；	曲线拟合结果精度较低。	
		2. 且曲线的函数方程简单。		

方法名称		优点	缺点	图示
分段矢量化法	固定步长法	曲线的拟合精度较高。	1. 曲线不连续且不闭合，需要添加竖直线段连接相邻曲线端点才能闭合； 2. 曲线函数方程复杂。	
	基于误差分析改进的固定步长法	曲线拟合精度高。		
	搜索步长法	1. 曲线拟合精度较高； 2. 曲线方程与固定步长法相比较为简单。		

六、参考文献

[1] 严素蓉, 朱桂林, 徐从富, 一种位图矢量化新方法, 计算机工程与应用, 41 (14): 85-87, 2005.

[2] 付丽琴, 陈树越, 一种改进的边界轮廓矢量化算法 [J]. 测试技术学报, 16 (4) :241-244, 2002.

[3] 张伟刚, 赖小勤, 点阵图形、图形矢量化算法研究, 广西工学院学报, 9 (2): 30-33, 1998.

[4] Muhammad Sarfraz. Some algorithms for curve design and automatic outline

## 参赛队号#1031

capturing of images[J]. International Journal of Image and Graphics, 4(2): 223-239, 2004.

数学中国提供 (www.madio.net)

## 附 录

FindPoints.m	脚本文件 1，首先运行本脚本提取有序轮廓点	
程序说明	提取图形轮廓并跟踪得到有序轮廓点	
包含函数文件	FindNeibour.m	Insertch.m
	InChPts.m	
	dis.m	
主要变量说明	OrderPts 有序轮廓点集	
<pre> clc clear Img=imread('位图.bmp'); Grey=Img(:,:,1);%I 就是灰度矩阵 %图像轮廓提取 BW1 = edge(Grey,'sobel');  %轮廓跟踪 %为后续计算方便点坐标用栅格行列(i,j)表达 %实际坐标应为(j,-i),并且此行列坐标小 0.5 %即最后实际坐标应为(j-0.5,-i+0.5) [ptx,pty]=find(BW1==1);%无序轮廓点集合 Spt=[ptx(1),pty(1)];%确定起点 Spt OrderPts=[Spt]; Cpt=Spt;Lpt=Spt;i=1;%初始化，Cpt 当前点，Lpt 上一点  %求跟踪目标 Visit=BW1; [N]= FindNeibour(BW1,Cpt,Lpt); while (Spt(1)~=N(1)    Spt(2)~=N(2))     Lpt=Cpt;     Cpt=N;     [N,Visit]= FindNeibour(Visit,Cpt,Lpt);     Visit(ptx(1),pty(1))=1;     if isempty(N)         break;     end     OrderPts=[OrderPts;N]; end TureOrder=[OrderPts(:,2),-OrderPts(:,1)];%将有序点转为实际坐标      break; end end </pre>		



```

if ~isempty(upl)&&~isempty(downl)
    tp1=InChPts(upl,:);
    tp2=InChPts(downl,:);
    if (dis(pt,tp1)>(Delt/2))&&(dis(pt,tp2)>(Delt/2))
        InChPts=Insertch(InChPts,downl,pt);
    end
end
end
end
i=i+Delt;
end

```

Trace.m	脚本文件 2，FindPoints.m 运行之后运行本脚本	
程序说明	去除冗余点，并提取轮廓特征点，最后利用原始轮廓点插值特征点	
包含函数文件	Nonred.m	
	Inchpts.m	
主要变量说明	NonredPts	去除冗余点后轮廓点
	ChPts	插值前特征点
	InChPts	插值后特征点

```

%去除冗余点坐标
tpts=[OrderPts(:,2),-OrderPts(:,1)];%有序坐标转换为实际坐标信息
tmpts=tpts;
NonredPts=[];
NonredPts=Nonred(tmpts);
%提取 SDRIC 点和长直线段端点
%求线段长度总和 sum
sumD=0;
n=length(NonredPts);
for i=2:length(NonredPts)
    dx=NonredPts(i,1)-NonredPts(i-1,1);
    dy=NonredPts(i,2)-NonredPts(i-1,2);
    sumD=sumD+sqrt(dx*dx+dy*dy);
end
%求  $\Sigma$ ,  $\lambda$  值
h=5;l=0;
Sigma=sumD/(n-1)+h;%h:0-5
Lambda=sumD/(n-1)-l;%l:0-1

i=3;
tmpts=NonredPts;
sp=tmpts(i-2,:);
mp=tmpts(i-1,:);

```

```

Dm1=0;
flag=0;%相对长直线段提取标记
ChPts=[];%特征点集合
while i<n
    tp=tmppts(i,:);
    fp=tmppts(i+1,:);
    D0=dis(sp,mp);
    D1=dis(tp,mp);
    D2=dis(fp,tp);
    delt=((sp(1)-mp(1))*(fp(1)-tp(1))>0)||((sp(2)-mp(2))*(fp(2)-tp(2))>0);
    if delt %提取 SDRIC 点
        ChPts=[ChPts;mp;tp];
        flag=0;
        Dm1=D1;
        sp=tp;
        mp=fp;
        i=i+2;
    else
        if D0>Sigma %提取长直线段
            ChPts=[ChPts;sp;mp];
            flag=0;
            sp=mp;
            mp=tp;
            Dm1=D0;
            i=i+1;
        else if (D0>Lambda*D1)&&(D0>Lambda*D2)&&(D0>Lambda*Dm1)&&(flag~=1)
            %提取某些相对长直线段
            flag=1;
            sp=mp;
            mp=tp;
            Dm1=D0;
            i=i+1;
        else %其他情况
            sp=mp;
            mp=tp;
            Dm1=D0;
            i=i+1;
        end
    end
end
end
end
ChPts= Unich(ChPts);%删除重复点
%加入起点终点
Start=NonredPts(1,:);%起点

```

```
End=NonredPts(end,:);%终点
ChPts=[Start;ChPts;End];
ChPts= Unich(ChPts);%删除重复点
ChPts=[ChPts;Start];

%特征点插值
epsilon=10;%0-10，微调  $\delta$  的正数值
Delt=ceil((Sigma+epsilon));
n=[];
n=length(TureOrder);
InChPts=ChPts;
i=Delt;
while i<n
    pt=TureOrder(i,:);
    if isempty(Inchpts(pt,InChPts))%pt 就是特征点,退出本次循环
        upl=[];
        downl=[];
        %往下遍历找到下特征点

        for j=i-1:-1:1
            tpt=TureOrder(j,:);
            %j tpt]
            l=Inchpts(tpt,InChPts);
            if ~isempty(l)
                downl=l;
                break;
            end
        end
        %往上遍历找到下特征点
        for j=i+1:n
            tpt=TureOrder(j,:);
            %j tpt]
            l=Inchpts(tpt,InChPts);
            if ~isempty(l)
                upl=l;
                break;
            end
        end
        if ~isempty(upl)&&~isempty(downl)
            tp1=InChPts(upl,:);
            tp2=InChPts(downl,:);
            if (dis(pt,tp1)>(Delt/2))&&(dis(pt,tp2)>(Delt/2))
                InChPts=Insertch(InChPts,downl,pt);
            end
        end
    end
end
```

end
end
i=i+Delt;
end

Vectorization.m	脚本文件 3，运行完脚本文件 1、2 后运行本脚本文件	
程序说明	对插值后特征点进行拟合得到曲线方程	
包含函数文件	FunVec.m	FunVec4.m
	FunVec2.m	
	FunVec3.m	
主要变量	P1 变换步长参数列表	
	P2 固定步长参数列表	
	P4 固定步长贝塞尔曲线参数列表	
<pre> %矢量化 clc %figure B0=3;%c 初始步长 %基于误差分析的变化步长 [ delt1,P1,xx1,yy1] = FunVec( InChPts,B0); %固定步长 [ delt2,P2,xx2,yy2] = FunVec2( InChPts,B0); %整体贝塞尔曲线 [ xx3,yy3] = FunVec3( InChPts); %固定步长贝塞尔曲线 [P4,xx4,yy4] = FunVec4( InChPts,B0); </pre>		

函数文件	
FindNeighbour.m	<pre> function [N,BW]= FindNeighbour(BW,C,L) H=zeros(3,3); 返回当前点 C    BW(C(1),C(2))=0; 的非上一点 L    BW(L(1),L(2))=0; 的邻域点 N      [m,n]=size(BW);                   N=[];                   %提取邻域点信息                   for i=-1:1                       for j=-1:1                           if i+C(1)&gt;0&amp;&amp;i+C(1)&lt;=m                               if j+(C(2))&gt;0&amp;&amp;j+C(2)&lt;=n                                   H=BW(C(1)-1:C(1)+1,C(2)-1:C(2)+1);                                   if BW(i+C(1),j+C(2))==1                                       N=[i+C(1),j+C(2)];                                   end                               end                           end                       end                   end </pre>

## 参赛队号#1031

	<pre> end end end </pre>
<b>InChPts.m</b> 返回点 pt 在点集中的位置，没有则返回空	<pre> function I=Inchpts( pt,Pts ) I=[];% for i=1:length(Pts)     if sum((Pts(i,:)==pt))==2         I=i;         break;     end end end end </pre>
<b>dis.m</b>	
<b>Insertch.m</b> 在点集 chpt 第 index 插入点 pt	<pre> function chpt= Insertch( chpt,index,pt) chpt1=chpt(1:index,:); chpt2=chpt(index+1:end,:); chpt=[chpt1;pt;chpt2];  end </pre>
<b>Nonred.m</b> 去除点集 tpts 中的冗余点，返回点集 NonredPts	<pre> function NonredPts= Nonred( tpts) tmpts=tpts; n=length(tpts); i=3; sp=tmpts(i-2,:); mp=tmpts(i-1,:); fp=tmpts(i,:); lfp=tmpts(i+1,:);%求斜率变化率 while i&lt;n-1;     k1=(mp(2)-sp(2))/(mp(1)-sp(1));     k2=(fp(2)-mp(2))/(fp(1)-mp(1));     k3=(lfp(2)-fp(2))/(lfp(1)-fp(1));     if k1== -inf         k1=inf;     end     if k2== -inf         k2=inf;     end     if k3== -inf         k3=inf;     end     if k1==k2%斜率相等         tpts(i-1,:)= [inf,inf];     end end </pre>

## 参赛队号#1031

	<pre> else if (k1==k3)&amp;&amp;(k2==0    k2==inf) %      tpts(i,:)=inf,inf]; %      tpts(i-1,:)=inf,inf]; %标记删除该点 else if k1*k2&gt;0 %斜率同向 %else if (k2-k1)*(k3-k2)&gt;=0%斜率变化率相同 %else if (k2-k1)*(k3-k2)&gt;0&amp;&amp;k1*k2&gt;0%斜率同向且斜率变化率相同 %else if k1*k2&gt;=0&amp;&amp;k2*k3&gt;=0&amp;&amp;(k2-k1)*(k3-k2)&gt;=0       tpts(i,:)=inf,inf];%标记删除该点     end   end end i=i+1; sp=tmpts(i-2,:); mp=tmpts(i-1,:); fp=tmpts(i,:); lfp=tmpts(i+1,:); end NonredPts=[]; for i=1:n     if tpts(i,1)&lt;inf         NonredPts=[NonredPts;tpts(i,:)];     end end end end </pre>
<p>FunVec.m</p> <p>搜索步长法拟合曲线方程</p>	<pre> function [ delt1,P,xx,yy] = FunVec( InChPts,B0) %变步长 P=[];%记录参数 tpts=[]; tpts=[InChPts;]; i=1; xx=[]; yy=[]; B=B0; while i+B&lt;length(InChPts)     B=B0;     x=tpts(i:i+B,1);     y=tpts(i:i+B,2);     k=polyfit( x,y,1);     Sig01=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值     %Sig01=inf;     k=polyfit( x,y,2);     Sig02=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值 </pre>

```

% Sig02=inf;
if Sig01<Sig02
    n=1;%选择一次多项式拟合
    Sig0=Sig01;
else
    n=2;%选择二次多项式拟合
    Sig0=Sig02;
end
tx=x;
ty=y;
if i+B+1<length(InChPts)
    B=B+1;
    tx=tpts(i:i+B,1);
    ty=tpts(i:i+B,2);
    k=polyfit( tx,ty,n);
    Sig=sum(abs(OLSCal(tx,k)-ty))/length(ty);
    while Sig<=Sig0&&(i+B+1<length(InChPts))
        x=tx;
        y=ty;
        B=B+1;
        tx=tpts(i:i+B,1);
        ty=tpts(i:i+B,2);
        k=polyfit( tx,ty,n);
        Sig=sum(abs(OLSCal(tx,k)-ty))/length(ty);
    end
% minx=min([tx(1) tx(end)]);
% maxx=max([tx(end) tx(1)]);
end
minx=tx(1);
maxx=tx(end);
if n==1
    k=polyfit( tx,ty,1);
    P=[P;i i+B minx maxx 0 k];
else
    k=polyfit( tx,ty,2);
    P=[P;i i+B minx maxx k];
end
end
i=i+B;
B=B0;
end
%拟合结尾
x=tpts(i:end,1);
y=tpts(i:end,2);
k=polyfit( x,y,1);

```

## 参赛队号#1031

	<pre> %Sig01=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值 k=polyfit( x,y,2); Sig02=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值 %      minx=max([x(1) x(end)]); %      maxx=max([x(end) x(1)]); minx=x(1); maxx=x(end); if Sig01&lt;Sig02     k=polyfit( x,y,1);     P=[P;i length(tpts) minx maxx 0 k]; else     k=polyfit( x,y,2);     P=[P;i length(tpts) minx maxx k]; end xx=[]; yy=[]; NeoP=P; for i=1:size(P,1)     tx=linspace(NeoP(i,3),NeoP(i,4),10);     ty=P(i,5)*tx.*tx+P(i,6)*tx+P(i,7);     xx=[xx tx];     yy=[yy ty]; end xx=[xx xx(1)];%连接首尾,闭合 yy=[yy yy(1)]; %计算误差 delt1=0; sumn=0; for i=1:size(P,1)     txx=lnChPts(P(i,1):P(i,2),1);     tyy=lnChPts(P(i,1):P(i,2),2);     tcy=P(i,5)*txx.*txx+P(i,6)*txx+P(i,7);     delt1=delt1+sum(abs(tcy-tyy));     sumn=sumn+length(txx); end delt1=delt1/sumn; end </pre>
<b>FunVec2.m</b> 基于误差分析 的固定步长曲 线拟合	<pre> function [ delt2,P,xx,yy] = FunVec2( lnChPts,B0 ) %固定步长 P=[];%记录参数 tpts=[]; tpts=[lnChPts;]; i=1; xx=[]; </pre>



```

yy=[];
B=B0;
while i+B<length(InChPts)
    B=B0;
    x=tpts(i:i+B,1);
    y=tpts(i:i+B,2);
    k=polyfit(x,y,1);
    Sig01=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值
    %Sig01=inf;
    k=polyfit(x,y,2);
    Sig02=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值
    %Sig02=inf;
    if Sig01<Sig02
        n=1;%选择一次多项式拟合
        Sig0=Sig01;
    else
        n=2;%选择二次多项式拟合
        Sig0=Sig02;
    end
    tx=x;
    ty=y;
    minx=tx(1);
    maxx=tx(end);
    if n==1
        k=polyfit(tx,ty,1);
        P=[P;i i+B minx maxx 0 k];
    else
        k=polyfit(tx,ty,2);
        P=[P;i i+B minx maxx k];
    end
    i=i+B;
end
%拟合结尾
x=tpts(i:end,1);
y=tpts(i:end,2);
k=polyfit(x,y,1);
Sig01=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值
k=polyfit(x,y,2);
Sig02=sum(abs(OLSCal(x,k)-y))/length(y);%将该值作为阈值
minx=x(1);
maxx=x(end);
if Sig01<Sig02
    k=polyfit(x,y,1);
    P=[P;i length(tpts) minx maxx 0 k];

```

	<pre> else     k=polyfit( x,y,2);     P=[P;i length(tpts) minx maxx k]; end xx=[]; yy=[]; NeoP=P; for i=1:size(P,1)     tx=linspace(NeoP(i,3),NeoP(i,4),10);     ty=P(i,5)*tx.*tx+P(i,6)*tx+P(i,7);     xx=[xx tx];     yy=[yy ty]; end xx=[xx xx(1)];%连接首尾,闭合 yy=[yy yy(1)]; %计算误差 delt2=0; sumn=0; for i=1:size(P,1)     txx=lnChPts(P(i,1):P(i,2),1);     tyy=lnChPts(P(i,1):P(i,2),2);     tcy=P(i,5)*txx.*txx+P(i,6)*txx+P(i,7);     delt2=delt2+sum(abs(tcy-tyy));     sumn=sumn+length(txx); end delt2=delt2/sumn; end </pre>
FunVec3.m 整体贝塞尔曲线拟合	<pre> function [ xx,yy ] = FunVec3( lnChPts ) %整体贝塞尔曲线 x = lnChPts(:,1);  y = lnChPts(:,2); [xx,yy]=bezier(x,y); end </pre>
FunVec4.m 分段贝塞尔曲线拟合	<pre> function [P,xx,yy ] = FunVec4( lnChPts,B0) % xx=[];yy=[]; B=B0; P=[]; tpts=lnChPts; i=1; while i+B&lt;size(lnChPts,1)     x=tpts(i:i+B,1);     y=tpts(i:i+B,2);     P=[P;i i+B x' y']; end </pre>

## 参赛队号#1031

	<pre> [tx ty]=bezier(x,y); xx=[xx tx]; yy=[yy ty]; i=i+B; end %拟合结尾 x=tpts(i:end,1); y=tpts(i:end,2); n=length(x); tp=zeros(1,2*(B-n+1)); P=[P;i length(tpts) x' y' tp]; [tx ty]=bezier(x,y); xx=[xx tx]; yy=[yy ty]; xx=[xx xx(1)];%连接首尾,闭合 yy=[yy yy(1)]; clc end </pre>
bezier.m 贝塞尔插值	<pre> function [X,Y]=bezier(x,y) % syms t % B=[(1-t)^3,3*t*(1-t)^2,3*t^2*(1-t),t^3]; n=length(x); t=linspace(0,1); xx=0;yy=0; for k=0:n-1     tmp=nchoosek(n-1,k)*t.^k.*(1-t).^(n-1-k);     xx=xx+tmp*x(k+1);     yy=yy+tmp*y(k+1); end X=xx; Y=yy; end </pre>
OLSCal.m	<pre> function y = OLSCal( x,k ) y=polyval(k,x); end </pre>