# 武漢理工大學

数学建模暑期培训论文

第 2 题

# 基于指派问题模型的指纹识别与匹配

第 1 组


| 姓名 | 方向 |
| --- | --- |
| 阮滨（组长） | 编程 |
| 王宇 | 建模 |
| 王家柯 | 写作 |


**2020 年 8 月 11 日**

# 摘要

本文首先通过图像分割、图像增强等方法对指纹图像进行了预处理，并结合全局特征点和细节特征点构建了具有旋转、平移不变性的指纹特征，之后通过特征编码将其以较少的字节数进行存储。此外，本文运用指标归一化后的欧氏距离对指纹特征点相似程度进行表达，进而以最大化最佳的特征点匹配方案下的指纹相似程度作为目标，将指纹匹配问题转化为指派问题进行求解。最后，运用方向场信息，对指纹的纹型进行分类。

由于附件中的指纹图像存在模糊、损坏等情况，故需要对指纹图像进行预处理。本文首先采用了**基于梯度向量模的图像分割算法**，并通过邻域分析消除了独立背景块。然后通过 **Gabor 滤波**对图像进行了增强，突出了指纹的脊线特征。考虑到全局阈值二值化方法对图像信息的破坏，本文采用了**结合方向信息的自适应动态阈值二值化方法**。最后采用了**改进的 OPTA 算法**对图像进行细化，使指纹的特征点信息更为明显。

针对问题一需要获得指纹的特征并进行编码。考虑到指纹图像可能存在的旋转、平移以及失真，本文通过**结合全局特征点和细节特征点，构建具有旋转、平移不变性的极坐标系**，增强特征的可靠性。此外，由于手指在按压的过程中，外围的指纹因皮肤的弹性受到拉伸，使得特征信息产生误差，故仅在中心点的邻域内选取特征点，进而本文选取了包括**极坐标、与中心点的方向场角度差、特征点种类等五个指标**。最后将其以**特征编码**的方式进行存储，并通过计算发现，在题目的编码字节的限制下，每个指纹最多可存储 22 个特征点。

针对问题二需要在上述编码的基础上衡量指纹的相似度。本文首先对问题一中每个细节特征点的五个指标进行了**归一化处理**，并通过**欧式距离**计算了特征点之间的相似程度。同时，考虑到相同指纹中各特征点的唯一对应关系，可以认为必须在两个指纹的特征点在达到整体最佳配对的前提下才可以合理的计算其相似程度。因此，本文将指纹匹配问题转化为**指派问题**进行分析，并以具有惩罚图像不完整性性质的相似性得分计算方法作为最大化目标，构建起**具有一定误差包容空间的整数规划模型**，进而选取了 4 组算例对模型进行验证，通过 LINGO 得到相似性得分分别为 **0.89，0.67，0.52，0.55**，基本符合主观验证的结果。

针对问题三需要对附件中的指纹图像进行对比分类。本文首先根据前文所求的**方向场信息**，运用 **Poincare Index 算法**得到指纹的全局关键点，并通过关键点中核心点和三角点的个数对指纹的纹型进行初步判断，再通过核心点和三角点连线上的方向角均值最终确定指纹纹型。最后通过与主观分析的结果进行对比，发现算法对于纹型的识别成功率可达 **81.25%**。

本文的优点为：1. 将复杂的指纹识别问题转化为经典的指派问题，极大简化了问题的复杂程度；2. 通过 0-1 变量的非线性项线性化，提高了模型的求解速度；3. 对指纹图像进行了充分的预处理，保证了后续分析的准确性。



**关键词：　指纹识别　　指派问题　　方向场　　模式识别**

# 目录

# 1、 问题重述

## 1.1 问题背景

随着计算机技术和人工智能的快速发展，生物特征因为其具有唯一性、稳定性和较高的防伪性等优势，广泛被用于身份识别领域。其中指纹识别与其他生物特征识别相比，不仅具有很高的稳定性，而且在实用性、便携性广泛被使用于社会安全、金融安全、公司考勤等方面。因此如何以最小的数据量提取指纹的特征，以及快速对指纹进行识别十分有必要。

## 1.2 问题重述

问题一：提取附件一所给指纹的特征，并用一种不超过 200 字节的方式刻画指纹的基本特征，并解释其数学原理。

问题二：利用问题一所给的指纹提取和编码方式，对附件中的指纹进行特征提取和编码，并给出比较指纹异同的方法。

问题三：利用问题二得到的比较分析方法，给出一个分类依据并对附件中的 16 个指纹进行对比和归类。

# 2、 模型的假设

- 假设图形的变化过程中仅发生旋转和放缩，不考虑镜像变化；
- 假设本文讨论的纹型均为数据库可查询的已知纹型；
- 假设特征点只考虑端点和交叉点；
- 假设特征点之间是相互独立的。

# 3、 符号说明

| 符号 | 意义 |
|---|---|
| $\partial(i,j)$ | 梯度向量模 |
| $\theta(m,n)$ | 块方向 |
| $I$ | 特征点种类 |
| $P_0(x_c,y_c)$ | 极点坐标 |
| $d_{ij}$ | 两点之间的欧式距离 |
| $M_{MO}$ | 匹配成功点的个数 |

注：其他未列出的符号以第一次出现时的解释为准

## 4、 数据的预处理

通过翻阅附件中的指纹图，可以很明显的看到未经过处理的指纹图存在着纹路不清晰，断点等问题。这些缺陷在之后的特征提取中会带来巨大的干扰，因此进行图像的处理十分有必要。下图为本文图片预处理思路图：


**图 1　指纹图预处理思路图**

### 4.1 基于梯度向量模的指纹图像分割

通过文献可知 [1]，指纹的前景区由脊线和谷线相间而成，灰度的变化较大，而背景区则会出现灰度变化不明显的特征。基于这个特征，如果用 Sobel 算子求取每个像素点的水平梯度和竖直梯度（$\partial_x, \partial_y$），然后将水平梯度和竖直梯度求和，从而得出一个新的向量特征，称之为向量梯度模 $[\partial(i,j)]$。对于指纹的图片中，位于指纹边缘处的梯度向量模的值相对于脊谷线中间部分和背景区域像素点的数值较小。

如果将一整幅图像划分成 $w \times w$ 大小的相互不折叠的小块，将小块内的所有像素点的梯度向量模作为一小块的特征，那么位于前景区域的小块特征值就会大于背景区域的小块特征值。基于这一特征，通过设定一个阈值，就可以完成对前景区域和背景区域的分割。分割算法具体步骤如下。

*Step*1 将指纹图像划分成 $w \times w$ 的互不重叠的小块。$w$ 应该大于脊线的平均宽度，通过反复实验，确认 $w = 5$ 较为合适。

*Step*2 利用 Sobel 算子计算附件中指纹图像的像素点 $i, j$ 的水平梯度灰度梯度 $[\partial_x(i,j)]$ 和竖直梯度灰度梯度 $[\partial_y(i,j)]$，计算公式如下

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

**图 2　Sobel 算子**

$$\begin{aligned}\partial_x(i,j) = &[N(i+1,j-1) + 2N(i+1,j) + N(i+1,j+1)] \\ &-[N(i-1,j-1) + 2N(i-1,j) + N(i-1,j+1)];\end{aligned} \tag{1}$$

$$\begin{aligned}\partial_y(i,j) = &[N(i-1,j+1) + 2N(i,j+1) + N(i+1,j+1)] \\ &-[N(i-1,j-1) + 2N(i,j-1) + N(i+1,j-1)].\end{aligned} \tag{2}$$

*Step*3 通过公式1，2得到的 $|\partial_x(i,j)|$ 和 $|\partial_y(i,j)|$ 计算各个像素点 $(i,j)$ 的梯度向量模 $|\partial(i,j)|$。公式如下

$$|\partial(i,j)| = |\partial_x(i,j)| + |\partial_y(i,j)|. \tag{3}$$

如果将其映射成为灰度图像，就会得到一幅 $L \times W$ 的梯度向量模图。

*Step*4 求各分块内所有像素点的梯度向量模的均值，并将其作为一个特征 $VM(m,n)$。第 $(m,n)$ 分块的梯度向量均值的计算公式如下：

$$VM(m,n) = \frac{1}{w \times w} \sum_{i=0}^{w-1} \sum_{j=0}^{w-1} |\partial(i + m \times w, j + n \times w)|. \tag{4}$$

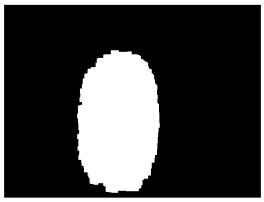
通过块梯度向量模的均值，指纹图像的前景区域和背景区域可以明显得被区分出来。

*Step*4 按照设定的阈值对指纹图像进行分割。分割规则如下：

$$S(m,n)) = \begin{cases} 1 & VM(m,n) \geq T \\ 0 & else \end{cases} \tag{5}$$

其中，1 代表前景区域，0 代表背景区域。

经过反复实验之后的效果对比，发现当取分割阈值 T=175 时，对附件中不同的指纹图像均能取得不错的分割效果。如下图：



(a) 指纹 1 分割图

(b) 指纹 2 分割图

**图 3　指纹分割图**

### 4.2 指纹图像增强

#### 4.2.1 指纹方向图的获取

根据相关文献可以发现，将指纹图像进行局部放大后，在一定范围内指纹的纹路是近似平行且具有一致的方向性，而且相互平行的纹路也具有固定的宽度和间隔。所以如果沿着垂直于纹路的方向来看，指纹的脊线、谷线像素点的灰度值大致形成一个二维的正弦波。因此，指纹纹路的分布具有较好的方向特性和频率特性。

指纹方向图，是抽象了指纹图像的上诉纹理的特性，以较为简化的形式直观反映指纹图像纹理的本质特征。现在最常用的方向图求取算法有梯度法和模板法。本文采用提复发求取方向图，算法步骤如下：

$Step1$ 将指纹图像划分成 $w \times w$ 的互不重叠的小块。通过反复实验，确认 $w = 5$ 较为合适。

$Step2$ 通过 $3 \times 3$Sobel 算子计算附件中指纹图像的像素点 $i, j$ 的水平梯度灰度梯度 $[\partial_x(i,j)]$ 和竖直梯度灰度梯度 $[\partial_y(i,j)]$。这里直接可以用前文得到的数据。

$Step3$ 计算每个 $w \times w$ 子块的块方向，第 (m,n) 子块的快方向计算公式如下：

$$\theta(m,n) = \frac{1}{2} tan^{-1}(\frac{V_x(m,n)}{V_y(m,n)}), \tag{6}$$

其中

$$\begin{cases} V_x(m,n) = \sum\limits_{i=0}^{w-1}\sum\limits_{j=0}^{w-1} 2\partial_x(i + m \times w, j + n \times w)\partial_y(i + m \times w, j + n \times w); \\ V_y(m,n) = \sum\limits_{i=0}^{w-1}\sum\limits_{j=0}^{w-1} [\partial_x^2(i + m \times w, j + n \times w) - \partial_y^2(i + m \times w, j + n \times w)]. \end{cases} \tag{7}$$

通常，经过公式7求出的 $\theta(m,n)$ 之后还要进行映射，因为块方向的范围应该为 $[0,\pi)$, 但是公式 7求得的方向 $\theta(i,j) \in [-\frac{\pi}{4}, \frac{\pi}{4}]$, 显然这不能表示出所有的块方向。所以我们需要将 $\theta(m,n)$ 从 $[-\frac{\pi}{4}, \frac{\pi}{4}]$ 映射到 $[0,\pi)$。映射公式如下：

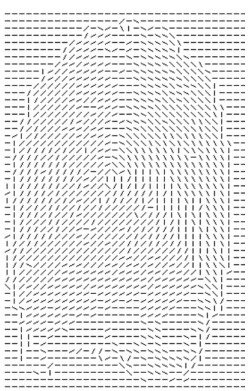
$$\theta(m,n) = \begin{cases} \theta(m,n) + 0.5\pi & V_y(m,n) < 0; \\ \theta(m,n) + \pi & V_x(m,n) < 0 and V_x(m,n) > 0; \\ \theta(m,n) & else. \end{cases} \tag{8}$$

由于有噪声、模糊、断裂等原因，会导致求得的方向如中存在方向凸块，因此可以利用小块邻域的方向信息进行平滑，修正由噪声引起的方向突变的子块的方向。

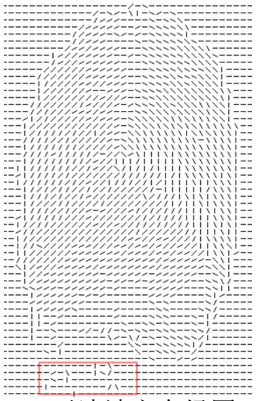
平滑处理的方法为：在使用公式6之前先用 $5 \times 5$ 大小的高斯低通滤波器对公式7求得的小块的 $V_x$ 和 $V_y$ 求取方块图 $\theta$，并进行范围映射。下图为方向场与原图叠加示意图

**图 4　方向场与原图叠加示意图**

下图为加滤波与不加滤波方向场图对比：可以看出滤波可以对红色方块中的信息进行修正。



(a) 有滤波方向场图



(b) 无滤波方向场图

**图 5　指纹方向场图**

### 4.2.2 指纹频率场的获取

由于前文提到的局部区域内指纹纹脊线和谷线交替出现，从沿着垂直于纹线的角度看，指纹的脊线和谷线上的像素点的灰度值形成一个类似正弦波的二维图形，而这个类正弦波的两个波峰和波谷的像素点个数的倒数则正好可以反映该局部区域内的脊线频率。频率场正好是有所有不互相折叠局域小块脊线组成的场结构。如图所示

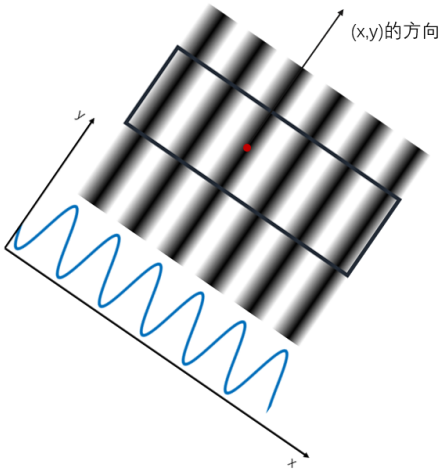



**图 6　频率场**

指纹图像频率场的获取需要以方向图的正确获取为基础，求取频率场的步骤如下：

*Step*1 将上一节分割成的 $w \times w$ 小块的中心点为中心，沿着垂直于脊线的方向，做一个长度为 $l$、宽度为 $w$ 的长方形窗口，并计算沿着脊线方向投射成的正弦波 $X[k]$，第 $(m,n)$ 块的中心点为 $(i,j)$，幅值 $X[k]$ 的计算公式如下。

$$X[k] = \frac{1}{w}\sum_{d=0}^{w-1} N(U,V) \quad k = 0,1,2,3...,l-1 \tag{9}$$

其中

$$\begin{cases} U = i + (d - \frac{w}{2})\cos O(m,n) + (k - \frac{l}{2})\sin O(m,n) \\ V = j + (d - \frac{w}{2})\sin O(m,n) - (k - \frac{l}{2})\cos O(m,n) \end{cases} \tag{10}$$

其中 $l = 15, w = 5$，U 和 V 是长方形窗口坐标系下的点，需要转换为图像坐标下的像素点进行计算。

*Step*2 在理想的情况下，像素点的灰度值沿着局部块方向上的投影近似于正弦波。然而现实情况中，由于噪声的存在，获取到投影分布图在大体上近似于正弦波，但是在细节处任然具有许多极值点，这些极值点会影响脊线频率的判断。如下图所示：

**图 7　沿脊线方向的灰度投影分布图**

为了去除噪声，需要对 $X[k]$ 信号进行平滑处理，假设噪声为高斯噪声的情况下用一个具有高斯包络的一维低通滤模板对信号进行卷积。选用滤波器 $[2,4,5,4,2]$ 进行平滑后的灰度投影如图所示，消除噪声后的投影分布基本接近正弦波形。

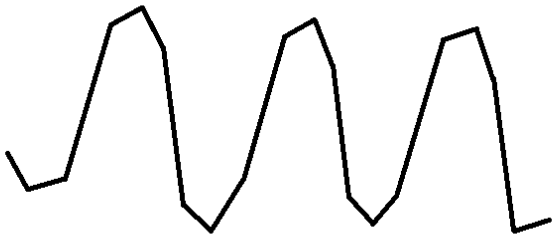


**图 8　平滑处理后的灰度投影分布图**

$Step3$ 计算这些极大值两点之间的平均值并求出频率，记平均值为 $T(m,n)$, 则波形的频率可表示为：$F(m,n)=\frac{1}{T(m,n)}$

如果 $X[k]$ 序列中没有明显的极大值和极小值，使得该块不能形成明显的正弦波，那么将该块的极限频率设置为 -1，表示该块的频率无效。另外，对于以固定分辨率扫描的指纹图像，局部邻域分块的脊线频率具有一定的范围，若频率的估计值超过这个范围则同样将频率设为 -1，表示频率无效。

$Step4$ 对于脊线频率被置为 -1 的块，其频率值需要根据相邻快递有效频率进行修正。修正方法如下：

$$F'(m,n)=\begin{cases}F(m,n) & F(m,n)\neq -1\\ \frac{\sum_{u=-\frac{w_g}{2}}^{\frac{w_g}{2}}\sum_{v=-\frac{w_g}{2}}^{\frac{w_g}{2}},W_g(u,v)\mu(F(m+u,N+v))}{\sum_{u=-\frac{w_g}{2}}^{\frac{w_g}{2}}\sum_{v=-\frac{w_g}{2}}^{\frac{w_g}{2}}W_g(u,v)\sigma(F(m+u,N+v))} & else.\end{cases} \tag{11}$$

其中

$$\begin{cases}\mu(x)=\begin{cases}0 & x\leq 0,\\ x & else;\end{cases}\\ \sigma(x)=\begin{cases}0 & x\geq 0,\\ x & else.\end{cases}\end{cases} \tag{12}$$

公式12中，$W_g$ 是个大小为 $w_g=7$ 的离散高斯核，它的均值、方差分别为 0 和 9。

经过以上四个步骤就可以求出所有分块的频率值，这些频率值则组成了该指纹的频率场。

### 4.3 基于 Gabor 滤波器的指纹增强算法

翻阅相关文献 [2]，可知由于增强指纹图像的目的在于消除纹线的断裂和粘连，减少这种噪声引起的伪特征。所以要想解决噪声带来的干扰，滤波器需要满足方向性、周期性、局部适应性、对称性等特点。

通过对文献的查找，发现 Gabor 滤波器具有良好的方向和频率选择性，能够较好的满足上述要求。Hong 等人在将 Gabor 滤波器应用在指纹图像增强的时候取得了较好的结果。

由于 Gabor 的特点以及对指纹图像对滤波的要求，一半设计为对偶的 Gabor 滤波器，它在空域中有如下形式

$$G(x,y,\theta,f_0)=exp-\frac{1}{2}(\frac{x_\theta^2}{\sigma_x^2}+\frac{y_\theta^2}{\sigma_y^2})\cos(2\pi f_o x_\theta) \tag{13}$$

其中

$$\begin{bmatrix}x_\theta\\y_\theta\end{bmatrix}=\begin{bmatrix}sin\theta & cos\theta\\-cos\theta & sin\theta\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} \tag{14}$$

公式17中 $\theta$ 为 Gabor 滤波器的方向，$f_0$ 是正弦平面波频率，$[x_\theta,y_\theta]$ 是笛卡尔坐标 $[x,y]$ 顺时针旋转 $(90°-\theta)$ 后的坐标，$\sigma_x$ 和 $\sigma_y$ 分别为沿着 $x$ 和 $y$ 轴的高斯包络常量 [3]。

为了将 Gabor 滤波器能用于指纹图像的增强，需要确立三个参数 $\theta$、$f_0$、$\sigma_x$ 和 $\sigma_y$。当 $\theta$ 与指纹局部纹线方向相同，正弦平面波的频率 $f_0$ 和局部的脊线频率相同时。Gabor 滤波器可以对指纹图像沿着脊线方向进行平滑处理。另外 $\sigma_x$ 和 $\sigma_y$ 的值取越大去噪能力越强，但是产生伪脊线的可能性越大。在这里取 1.6 与 1.6 时较为理想。

由于前文中分别求出了指纹图像的块方向图和块脊线频率场，所以 Gabor 滤波器方向 $\theta$ 和频率 $f_0$ 就可以确定了。增强后的图像通过下式就可以获取：

$$E(i,j)=\begin{cases}255 & S(i,j)=0\\ \sum_{u=-\frac{m}{2}}^{\frac{m}{2}}\sum_{v=-\frac{m}{2}}^{\frac{m}{2}}G(u,v,O(i,j),F(i,j))N(i+u,j+v) & S(i,j)\neq 0\end{cases} \tag{15}$$

其中，G 为 Gabor 滤波器，m 是滤波器的宽度，O 是指纹图像的快方向图，F 是指纹图像的频率场，N 是归一化之后的指纹图像，S 是指纹图像分割标记模板。滤波器的宽度 m 选取较为重要，宽度过大可能会在弯曲处破坏纹理结构，而宽度过小则难以达到去噪声的效果。本文通过对比观察确定滤波器宽度 m=5。

利用 Gabor 进行滤波时，首先根据块的方向和脊线频率生成针对的 Gabor 滤波器，然后将 $m\times m$ 大小的滤波器窗口的中心依次划过该块的每个像素点，并对这些点进行卷积计算，最后得到的值即为窗口中心像素点增强后的灰度值。

本文采用 matlab 对 Gabor 滤波器进行实现，以下是不同指纹的增强结果。

(a) 指纹 1 增强图　　(b) 指纹 2 增强图

**图 9　指纹增强图**

### 4.4 基于方向信息的指纹图像二值化方法

通过前文进行图像增强之后我们发现增强后的图像脊线谷线的对比非常明显，此时直接基于灰度进行二值化就可以取得比较满意的结果。但如果增强后的指纹图像脊谷线对比不太明显，部分纹路任然存在粘连情况时，则需要利用指纹的方向信息进行处理。

由于脊线的走向具有近似一致的方向性。因此结合方向特征进行二值化，不仅可以进行二值化处理，而且可以对纹线做进一步的修复。基于脊线的某一像素点来说，沿着脊线方向的像素点都应该是灰度值比较小的黑点，灰度值的和应该偏小；而垂直于脊线方向的像素点应该会存在灰度值较大的白点，其灰度值和偏大[4]。

利用这一特性，可以采用如下步骤进行二值化：

*Step*1 利用上文求得的指纹方向图，结合离散的八方向模板，计算沿着纹线方向的 m 个相邻像素点的灰度和 H，本文取 m=11。

*Step*2 计算沿着垂直于纹线方向的 m 个与之相邻的像素点灰度和，记作 V。

*Step*3 比较 H 和 V，判断该点是脊线上的点还是谷线上的点。如果 H<V 则该点在脊线上，将灰度值设置为 0，反之则设为 255。遍历所有像素点。

$$B(i,j)=\begin{cases}0 & H<V;\\ 1 & H\geq V.\end{cases} \tag{16}$$

最后二值化如下图所示：

(a) 指纹 1 二值化图　　(b) 指纹 2 二值化图

**图 10　指纹二值化图**

通过以上处理方式，我们可以发现此方法可以有效的去除脊线上的粘连、修复脊线的断裂。

### 4.5 OPTA 细化算法

由于二值化处理过后，指纹图像被分割为具有一定宽度的黑色脊线和白色谷线。然而在特征提取的过程中我们只需要脊线的结构特征，脊线的宽度反倒会对特征提取产生影响。因此为了提高特征提取的准确性，需要对二值化的指纹图像进行细化处理。

OPTA 细化算法是一种模块匹配的细化算法，它抽取与每个像素点相邻的 10 个像素，并与 8 个消除模板和两个保留模板进行比较，判断中心像素应该删除还是保留。以此实现了图像的细化处理 [5]。

图中左上角的 3 × 3 区域为消除模板的比较区域，构建的 8 个消除模板结构如下图所示，其中“1”代表目标点，“0”代表背景点，“×”代表目标点或者背景点，用整个 4 × 4 区域构建 6 个保留模板如图12所示。

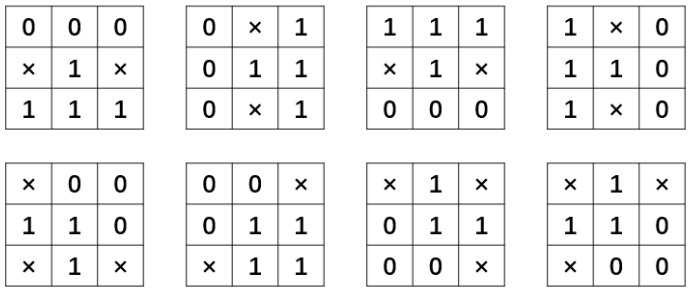


**图 11　改进的 OPTA 细化算法的消除模板**

OPTA 细化算法的具体步骤如下：

遍历二值化的指纹图像的每个像素，并选取其相邻的 15 个像素点，如上图所示将其 8 个邻域分别与上图 8 个消除模板进行比较，如果有一个模板可以与其匹配，则将抽取 15 个像素点与六个保留模板如图12 ，进行比较，如果与任意模板匹配，则保留该像素点，反之则删去。

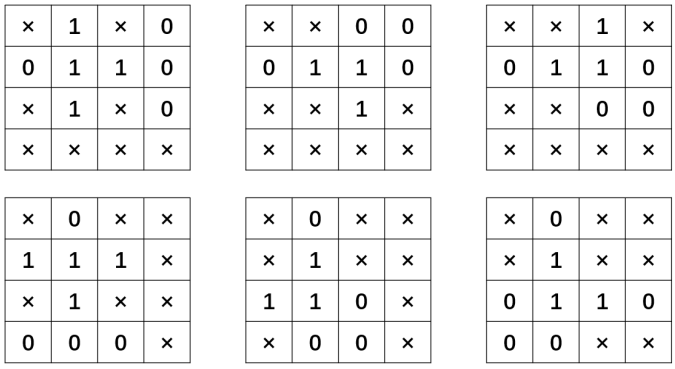


**图 12　改进的 OPTA 细化算法的保留模板**

以下是细化后的指纹图像。

(a) 指纹 1 细化图　　(b) 指纹 2 细化图

**图 13　指纹细化图**

# 5、 问题一的模型建立与求解

## 5.1 问题分析

在实际指纹检测的过程中，往往会出现指纹按压力度不均匀、指纹图像受损以及采集角度不同等情况。因此指纹图像的特征需要能够消除旋转、平移带来的影响并要求具有较强的鲁棒性。为此，本文拟采取了全局特征点和局部特征点结合的方式，建立起具有旋转、平移不变性极坐标系，并以此为基础确定特征集合。

而对于指纹特征的编码，拟采用特征向量的形式。由于特征集合的构成较为精简，因此本文采用此编码方式可以在给定字节限制的前提下存放尽可能多的特征信息。

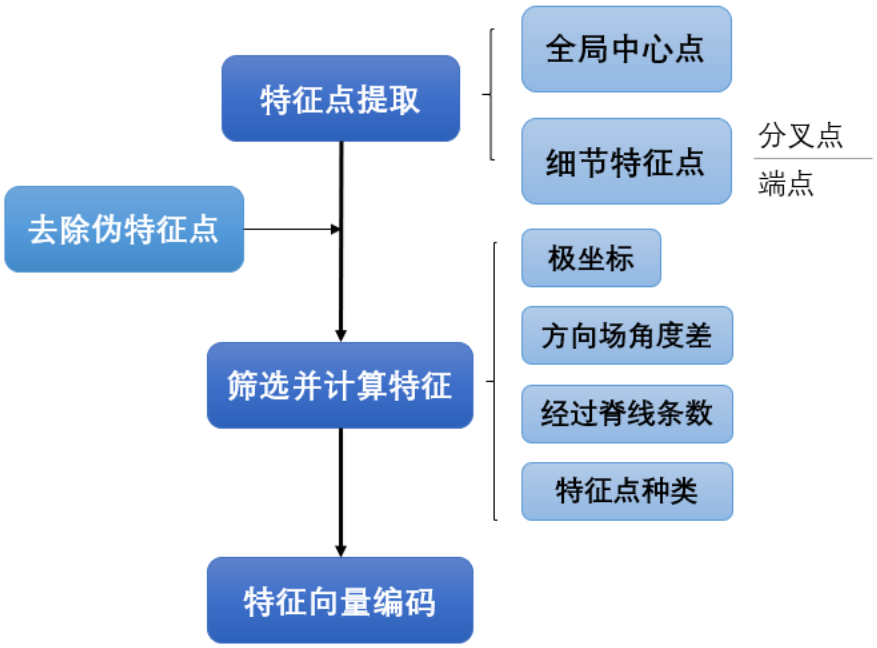



**图 14　问题一思路图**

### 5.2 建立基于指纹局域结构特征的编码模型

#### 5.2.1 基于曲率场的伪中心点选取

指纹的曲率场表示的是沿着固定长度的弧度，脊线方向的变换量，一般通过跟踪细化后的指纹极限获得。在指纹的某些区域（例如核心点区域）曲率值的变换遍布整个指纹图像。Yager et al. 利用三次薄板样条法表示每条细化后的脊线，从而计算出某些像素点的切线方向上的变换率，最后使用高斯滤波器来获取整张指纹图像的脊线曲率图。

薄板样条法：从细化后的图片中，获取离散的脊线。每条脊线用一个坐标集合表示，$(x_i, y_i)$ 和 $(x_{i+1}, y_{i+1})$ 表示离散脊线上相邻的两个像素的坐标位置。根据曲率的定义，第 $i$ 个像素点的曲率值的计算如下：

$$\begin{aligned} Curve_i = 4[(x_{i+k} - x_{i-k})(y_{i-k} - 2y_i + y_{i+k}) - (y_{i+k} - y_{i-k})(x_{i-k} - 2x_i + x_{i+k})] \\ \div[((x_{i+k} - x_{i-k})^2 + (y_{i+k} - y_{i-k})^{\frac{3}{2}})] \end{aligned} \tag{17}$$

其中，$k$ 表示步长。通过计算离散脊线上的每一个像素点的曲率值后，进行高斯低通滤波的平滑和插值，获得指纹前景区域的整个曲率场。

基于场方向的领域插值法：首先引入空间曲线曲率的计算通用公式。

$$k = lim_{\Delta S \to 0} \frac{\varphi}{\Delta S} \tag{18}$$

公式表明：曲线在 A 点处的曲率是 $\Delta S$ 趋于零时的极限，曲率 k 表示曲线从 A 点移动到 M 处时，切线方向上的平均变化率。根据此公式，将我们将图片按照上文分成大小为 $w \times w$ 的若干块，利用计算好的方向场计算每一块的曲率，中心块的曲率是与中心块方向场指向一致的两个领域块的方向场的角度差一半。如果中心块表示为 $B_k$，则此块曲率的计算公式如下：

$$B_k = \frac{|O(B_1) - O(B_2)|}{2} \tag{19}$$

这里，$B_1$ 和 $B_2$ 就是与中心点 $B_k$ 的方向场一致的两个块。这个方法计算曲率图更为高效，适用范围较广。将算出来的曲率最大的点记为中心点。

#### 5.2.2 特征点的选取

（1）指纹特征端点的提取

指纹特征端点是指纹纹线的两头末端，即终点和起点。在纹线像素模型的九点图中，如果中心点黑色方块是端点那么该黑方块的上下左右点显然只有两个，由于白点块的像素灰度值为 255，那么端点周围 8 个点两两相邻的两点差的绝对值之和就是

$2 \times 255$, 满足以此标准即确定中心黑色方块为端点。下图为在八邻域的所有状态中满足特征端点特征条件的八种情况 [6]。

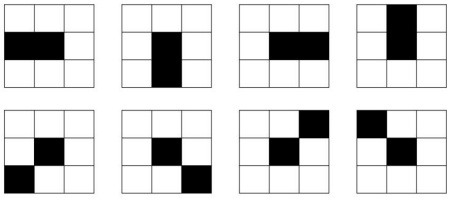


**图 15　端点模板**

提取方法为扫描某个点，如果周围 8 个点两两相邻的两个点的差的绝对值和为 $2 \times 255$ ，则为端点。

（2）指纹特征分叉点的提取

对于指纹特征的选取我们只要选取指纹的特征分叉点是两条纹线交会为一条纹线的交汇点。在纹线分叉点像素模型的九点图中，如果中心黑色方块点是分叉点，那么去掉分叉点后，纹线剩下三个黑色方块，每个黑色方块上下相邻的白色方块有两个，由于白色方块的像素值为 255，那么分叉点周围 8 个两两相邻的两个点差的绝对值之和就是 $6 \times 255$, 满足此标准即确定中心黑色方块为分叉点。下图为在八邻域的所有状态中满足特征端点特征条件的八种情况。

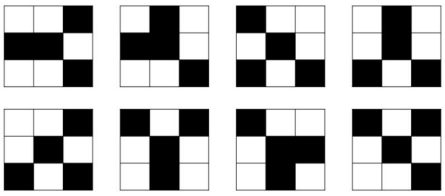


**图 16　分叉点模板**

提取方法为，扫描某个点，如果周围如果周围 8 个点两两相邻的两个点的差的绝对值和为 $6 \times 255$ ，则为分叉点。以下为特征点提取后的图片，其中端点用绿色圈出，分叉点用红色圈出。

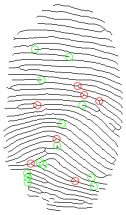
### 5.2.3 基于局部结构和细节特征点的指纹编码

1. 细节特征点的筛选

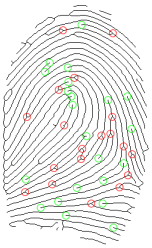
考虑到编码的字节限制，故有必要对细节特征点进行筛选。另外，由于皮肤的弹性，越靠近边缘的指纹越容易受到非线性扭曲的影响，因此本文仅选取临近中心点的部分细节特征点。

本文设定阈值 $R$, 并认为细节特征点与中心点 $P_0(x_0, y_0)$ 的距离小于此值时才纳入编码。进而通过遍历所有细节特征点，结合上述的特征计算方法，可以得到相应的特征集合 $S$，可表示如下：

$$S = (x_i, y_i, \theta_{fi}, I_i, n_i) | \sqrt{(x_i - x_{0i})^2 + (y_i - y_{0i})^2} \leq R, I_i \in a, b, c, i = 1, 2, ..., N \qquad (20)$$



(a) 指纹 1 特征点



(b) 指纹 2 特征点

**图 17　指纹细化图**

其中 $I$ 表示细节特征点的种类，若 $I=0$, 表示该特征点为中心点，$I=1$ 则表示该特征点为端点，$I=2$ 则说明该特征点为分叉点，$n$ 为中心点与特征点连线穿过的脊线数，$\theta_f$ 为特征点所在区域的块方向场角度。

需要注意的是，为保证满足字节限制的要求，若在半径为 $R$ 的领域内存在过多的特征点，本文仅选取距离中心点最近的 30 个特征点。

2. 构建极坐标系

将中心点 $P_0(x_c,y_c)$ 作为极点，其所在方向场的角度为 $\theta_c$，沿此角度作射线 $oP_0$ 作为极轴，构建具有旋转平移不变性的极坐标系。

3. 坐标系转换

为将上述的特征集合映射至构建的极坐标系中，需计算出特征点坐标与中心点的坐标以及两点所在方向场的方向差。

$$\begin{cases} \rho = \sqrt{(x-x_0)^2+(y-y_0)^2} \\ \theta = arctan\dfrac{y-y_0}{x-x_0} - \theta_c \\ \Delta\theta = \theta_f - \theta_{0f} \end{cases} \tag{21}$$

故特征集合修改为：

$$S=\{(\rho_i,\theta_i,\Delta\theta_i,I_i,n_i)|\rho_i \le R, I_i \in \{0,1,2,3\}, i=1,2,...,N\}$$

其中 $\theta, n, I$ 的含义与上文相同。

4. 特征编码本文采取特征向量的方式进行编码，即使得一个特征点的各特征作为一列内的元素。同一个指纹中的 $N$ 个特征点便对应有 $N+1$ 列向量，其中第一列为中心点的坐标，最后整合存储。

其中, $\rho$ 存放特征点的极长，占用 2 个字节; $\theta$ 存放特征点的极坐标角度，占用 2 个字节; $\Delta\theta$ 表示特征点方向场方向与中心点方向场方向的差值，占用 2 个字节; $I$ 表示特征点的种类，占用 1 个字节; $n$ 表示穿过的脊线数量，占用 2 个字节; 一个特征点占用 9 个字节; $N$ 个特征点，占用 9n 个字节; 综上，一个指纹最多存放 22 个特征点，即 198

字节，满足题目要求。排列方式如下：

| $\rho_1$ | $\theta_1$ | $\Delta\theta_1$ | $I_1$ | $n_1$ | $\rho_2$ | $\theta_2$ | $\cdots$ | $n_2$ | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 1 | 2 | 2 | 2 | | 2 | |

# 6、 问题二模型的建立与求解

## 6.1 问题分析

根据上文确定的特征选取方法，在此主要解决如何基于向量编码所提供的信息进行指纹的匹配。文本首先拟采用欧式距离衡量任意两个指纹之间的相似程度，并构建起指纹相似距离矩阵。结合相同指纹特征点一一对应的性质，易将此问题转化为指派问题进行分析。

在模型的建立过程中，还需要考虑到相似性的衡量方法以及较小误差的容错空间。对于前者，本文采用了 Ravi 基于惩罚图像不完整性的匹配分数计算方法 [7]；而对于后者，本文拟通过 0-1 变量的约束使得唯有两指纹差异大于某一阈值时才认定匹配失败。

## 6.2 指纹编码结果

以附件中的图一为例，其指纹的编码结果为：44.283 -1.510 0.748 1 4 46.872 -1.365 0.748 1 5 51.740 -1.236 0.805 1 6 19.026 -1.901 -0.459 1 0 13.928 -1.586 -0.286 1 1 28.792 -0.737 -0.435 1 2 65.521 -0.661 -0.616 1 6 71.868 0.818 -1.634 1 5 81.988 0.664 -2.331 1 6 34.132 -1.865 -0.358 2 0 96.773 -0.143 -0.692 2 12 39.051 -0.615 -0.499 2 3 51.225 0.513 0.760 2 4。

剩余的指纹编码结果见附录 A。

## 6.3 基于指派问题的指纹异同分析模型

考虑到理想的完全匹配状态下，两个指纹上的细节特征点应一一对应。但实际上，指纹采集质量或是指纹来源不同都有可能使得指纹的细节特征对应关系被破坏 [8]。

若在计算相似程度时，将两个指纹上的细节特征点错配、重复配，则会导致相似程度的计算出现较大误差 [9]。为此，本文通过将指纹的细节匹配问题转化为指派问题进行分析，使得在最佳匹配关系的基础上判断指纹相似程度。故首先需要计算任意两个指纹之间的相似性，并汇总为相似性矩阵。

两指纹之间的相似性通过如下方式定义：

Step1: 将各特征点在上述相对极坐标系中的坐标 $(\theta_i, \rho_i)$, 穿过脊线的条数 $n_i$，特征点与中心点的方向场角度差 $\Delta\theta_i$，特征点的种类 $I_i$ 都进行归一化。以特征点与中心点穿过脊线的条数为例：

$$n_k' = \frac{n_k - \min(n_i)}{\max(n_i) - \min(n_i)}$$

Step2: 运用欧氏距离计算任意两个指纹之间的差异性。

$$d_{ij} = \sqrt{(\theta_i - \theta_j)^2 + (\Delta\theta_i - \Delta\theta_j)^2 + (\rho_i - \rho_j)^2 + (I_i - I_j)^2 + (n_i - n_j)^2}$$

Step3: 构建相似性矩阵

进而构建指派问题的整数规划模型, 首先选取 Ravi 提出的匹配分数计算方法作为目标函数:

$$\max Score = \frac{M_{MO}}{\max(N_1, N_2)}$$

其中 $M_{MO}$ 为匹配成功的点的个数，$N_1$ 录入细节点的总数目，$N_2$ 模板细节点的总数目。这种匹配分数的计算方法在一定程度上惩罚了不完整指纹图像匹配以及重叠部分较少的情况。

根据特征点的一一对应原则建立相应的约束:

$$\begin{cases} \sum_{i=1}^{N_1} x_{ij} \leq 1 \quad , j = 1, 2, ..., N_2; \\ \sum_{j=1}^{N_1} x_{ij} \leq 1 \quad , i = 1, 2, ..., N_1. \end{cases} \tag{22}$$

实际匹配过程中允许存在一定误差，故两点之间相差距离 $d_{ij}$ 超过阈值 $L$ 时才认定匹配失败：

$$(1 - Z_{ij}) \geq X_{ij} d_{ij} - L \quad i = 1, 2, ..., N_1, j = 1, 2, ..., N_2. \tag{23}$$

通过 0-1 变量的约束得到成功匹配的特征点总数：

$$M_{MO} = \sum_{i=1}^{N_1} \sum_{j=1}^{N_1} Z_{ij} X_{ij}. \tag{24}$$

为提高求解速度，本文将上式线性化，即通过 (25) 代替 (24)：

$$\begin{cases} M_{MO} = \sum_{i=1}^{N_1} \sum_{j=1}^{N_1} p_{ij} \\ p_{ij} \leq X_{ij}, \quad i = 1, 2, ..., N_1, j = 1, 2, ..., N_2; \\ p_{ij} \leq Z_{ij}, \quad i = 1, 2, ..., N_1, j = 1, 2, ..., N_2; \\ Z_{ij} + x_{ij} \leq p_{ij}, \quad i = 1, 2, ..., N_1, j = 1, 2, ..., N_2. \end{cases} \tag{25}$$

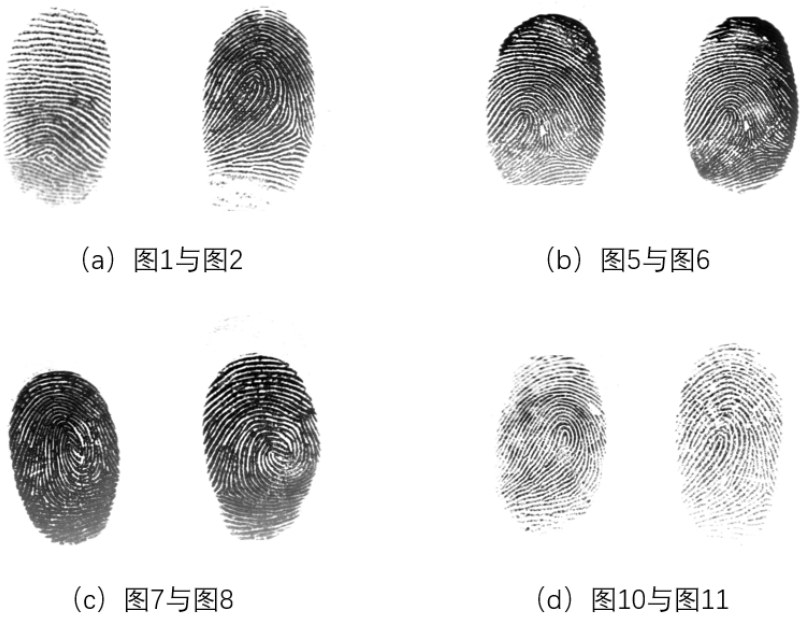
综上，将模型汇总可得到：

$$\max Score = \frac{M_{MO}}{\max(N_1, N_2)}$$

$$\begin{cases} \sum_{i=1}^{N_1} x_{ij} \le 1 \quad , & j = 1, 2, ..., N_2; \\ \sum_{j=1}^{N_1} x_{ij} \le 1 \quad , & i = 1, 2, ..., N_1; \\ (1 - Z_{ij}) \ge X_{ij} d_{ij} - L & i = 1, 2, ..., N_1, j = 1, 2, ..., N_2; \\ M_{MO} = \sum_{i=1}^{N_1} \sum_{j=1}^{N_1} p_{ij}; & \\ p_{ij} \le X_{ij} & i = 1, 2, ..., N_1, j = 1, 2, ..., N_2; \\ p_{ij} \le Z_{ij} & i = 1, 2, ..., N_1, j = 1, 2, ..., N_2; \\ Z_{ij} + x_{ij} \le p_{ij} & i = 1, 2, ..., N_1, j = 1, 2, ..., N_2; \\ Z_{ij}, x_{ij}, p_{ij} = 0 or 1 & \end{cases} \tag{26}$$

### 6.4 指派问题的模型求解

本文选取了附件中的 01.tif 与 02.tif、05.tif 与 06.tif、07.tif 与 08.tif，以及 10.tif 和 11.tif 作为 4 组算例，并基于 LINGO 编程实现了上述模型。



（a）图1与图2　　（b）图5与图6

（c）图7与图8　　（d）图10与图11

**图 18　指纹对比示意图**

**表 1　0.7 阈值下的对比得分**

| 指纹 5 和指纹 6 | 指纹 1 和指纹 2 | 指纹 7 和指纹 8 | 指纹 10 和指纹 11 |
|---|---|---|---|
| 0.789 | 0.556 | 0.552 | 0.478 |

结合主观分析，发现对于较为相似的图 7 与图 8，模型得分较高；而对于差异较大的图 10 和图 11，模型得分较低。与此同时，我们也注意到都为螺旋形的图 7 和图 8 以及都为环形的图 1 和图 2 显然具有一定相似性，但在得分上未能有明显区别，故接下来进行结果分析，对阈值参数的选取进行讨论。

### 6.5 结果分析

结合灵敏度分析的基本思想，本文以 0.1 为步长，初选阈值 $L = 0.7$ 的领域内选取了其它 5 个阈值，再次对上述的四个算例进行求解。结果如下：

**表 2　不同阈值对比分析图**

| | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|
| 指纹 5 和指纹 6 | 0.632 | 0.789 | 0.789 | 0.895 | 0.895 | 0.895 |
| 指纹 1 和指纹 2 | 0.389 | 0.444 | 0.556 | 0.667 | 0.722 | 0.722 |
| 指纹 7 和指纹 8 | 0.345 | 0.438 | 0.552 | 0.552 | 0.586 | 0.655 |
| 指纹 10 和指纹 11 | 0.364 | 0.409 | 0.478 | 0.545 | 0.545 | 0.545 |

可以发现当阈值降低时，模型整体分数降低；反之升高。而在模型的设定中，距离超过阈值时，则认为匹配失败，故与观察到的现象相符。

此外，对于过高或过低的阈值，都出现了指纹得分差异不明显的情况，使得指纹之间的相似性衡量准确性降低，因此在给定算例的前提下，本文给出如下公式调整阈值 $L$：

$$L = argmax(\sigma(L, K)).$$

其中 $K$ 为选定的算例集合。

在上述算例以及讨论的阈值 $L$ 中，$L = 0.8$ 为最优参数。

# 7、 问题三模型建立与求解

## 7.1 问题分析

针对附件中给的 16 张指纹图片，通过对指纹图像的处理，得到指纹分类所需要的信息，建立一个基于中心点和三角点这些特征点的指纹分类算法。

## 7.2 基于特征点的指纹分类模型

### 7.2.1 三角点和中心点的检测

由于中心点周围纹线呈半圆趋势，三角点周围的纹线由三个部分组成，而每个部分都呈现双曲线形状这些固有的几何特点，在本文中通过计算像素点周围的 Poincare 值来得到三角点和中心点，如 Poincare 值为 $\frac{1}{2}$, 则此处得到一个 core 点，若 Poincare 值为 $-\frac{1}{2}$, 则此处得到一个 delta 点。

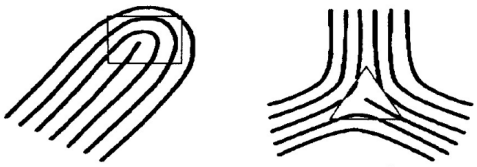


**图 19　core 和 delta 端点示意图**

（1）core 点的检测：

设 $(i,j)$ 为当前子图像的中心像素，$(i-1,j)$、$(i-1,j+1)$、$(i,j+1)$ 分别为其相邻的子图像的中心，假设 $O(i,j)$ 为 $(i,j)$ 处的方向值，令 $X_x(.)$ 和 $X_y(.)$ 分别为具有 N 个中心像素的闭合数字曲线的 x 与 y 坐标值，此处 N=4。现在顺次求每个 $(i,j)$ 处的 Poincare 值 [10]，即以逆时针方向求调整后的这个 N 子图像中心的方向信息差值的累积和：

$$Poincare(i,j)=\frac{1}{2\pi}\sum_{i=0}^{N}\Delta(k) \tag{27}$$

其中

$$\Delta(k)=\begin{cases}\partial(k) & |\partial(k)|<\frac{\pi}{2}\\ \pi+\partial(k) & |\partial(k)|\leq-\frac{\pi}{2}\\ \pi-\partial(k) & else\end{cases} \tag{28}$$

$$\begin{aligned}&\partial(k)=O(\Psi_x(i')\Psi_y(i'))-O(\Psi_x(i)\Psi_y(i)).\\&i'=(i+1)modN.\end{aligned} \tag{29}$$

若 $Poincare(i,j)$ 等于 $\frac{1}{2}$, 则此点为 core 点。

（2）delta 点的检测：

假设 $(i,j)$ 为当前子图像中心像素，$(i-1,j)-1$、$(i-1,j+1)$、$(i+1,j-1)$、$(i+1,j+1)$ 分别为与其相邻子图像的中心像素，同上沿着此四点求取各 $(i,j)$ 处的 Poincare$(i,j)$ 等于 $-\frac{1}{2}$ 则此点检测到一个 delta 点。

得到 core 点和 delta 点的数目后，即开始进行分类，这里采取有反馈的指纹分类方法，首先检测 core 点和 delta 点的对视，若 core 点和 delta 点对数大于 2，则对图像进行一次平滑，然后再分类（因为几乎没有 core 点或者 delta 多于 2 个的模型），否则直接按 core 点和 delta 点的个数与其相对位置进行分类。

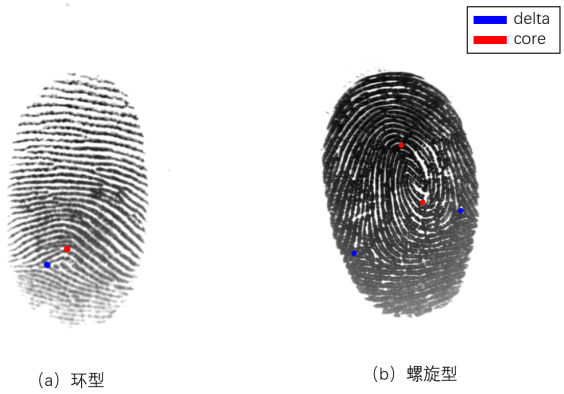



**图 20　特征点示意图**

### 7.2.2 分类原则

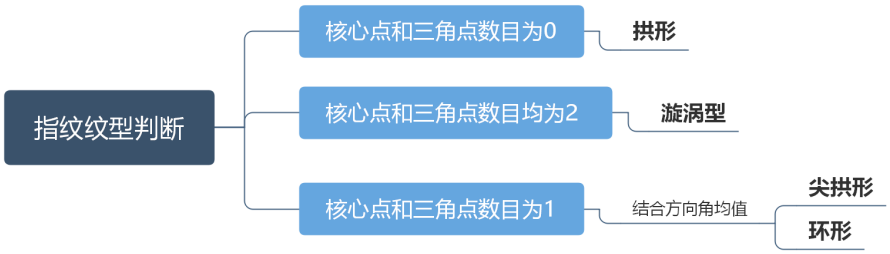



**图 21　指纹纹型判断示意图**

（1）若 core 点和 delta 点的数目为 0，则认为是拱形。

（2）若 core 点和 delta 点的数目为 2，则认为是螺旋形。

（3）若 core 点和 delta 点的数目为 1，则可能是拱形或者是环形，于是利用以下方法进行进一步的确认。

连接 core 点和 delta 点，在拱形指纹中，直线方向与实际方向一致，而在环形指纹中，直线却穿过指纹纹线。假设 $\beta$ 为 core 点和 delta 点连线的方向角，$\alpha_1,\alpha_2,\alpha_3,...,\alpha_n$ 为此线段上各段在指纹方向信息图上的方向角，如果平均值 $\frac{1}{n}\sum_{i=1}^{n}\sin(\alpha_i-\beta)$ 小于经验值 0.2，则认为该指纹为拱形，反之则为环形。

### 7.3 指纹分类模型的求解

利用 matlab 求解指纹图的 core 点和 delta 点个数后，机器自动识别了指纹的种类。通过查阅文献将自动识别与人工比对进行对比并判断正确率，如下表：

**表 3　识别对比**

| | 算法识别结果 | | 主观判断结果 | | 准确率/% |
|---|---|---|---|---|---|
| | 个数 | 占比/% | 个数 | 占比/% | |
| 环形 | 10 | 62.50% | 11 | 68.75% | 90.91% |
| 螺旋形 | 1 | 6.25% | 3 | 18.75% | 33.33% |
| 拱形 | 2 | 12.50% | 2 | 12.50% | 100% |
| 尖拱形 | 1 | 6.25% | 0 | 0 | / |
| 拒绝识别 | 2 | 12.50% | 0 | 0 | / |

综上，若以附件一中的指纹图像作为识别对象，该方法对于纹型的识别成功率可达 81.25%。

# 8、 模型总结与评价

## 8.1 模型优点

1. 在问题二中将指纹识别问题转化为了经典的指派问题，化繁为简。

2. 在问题一中考虑到手指皮肤的弹性造成扭曲，因此在特征点提取时，选取了受影响的较小的中心部分，故结果可靠性较高。

3. 在图像预处理中采用了结合方向信息的自适应动态阈值二值化方法，比传统的全局阈值二值化方法效果更好。

## 8.2 模型缺点

1. 由于分析的指纹图像数量较少，因此在模型参数的选择上难免有局限性。

2. 本文针对纹型分类的模型对于指纹图像的要求较为苛刻，若关键点被损坏则易发生拒认的情况。

## 8.3 模型改进

近年来，随着人工智能及相关技术的发展，学术界中提出了许多基于深度学习的指纹识别技术。在大量数据的加持下，指纹识别的准确率得到了极大的提升。

然而，深度学习算法的实施需要大量的数据。由于时间关系，未能额外搜寻到质量较高且被标记的学习样本。若将来时间允许，可以尝试开展相关的算法实施，进一步提升指纹识别的准确率

# 参考文献


[1] 郭玉兵. 指纹图像预处理算法研究 [D]. 山东大学, 2011.

[2] 李海燕, 程龙, 宗容, 等. 基于三方向图的多尺度平滑指纹奇异点检测 [J]. 华中科技大学学报 (自然科学版), 2019(3).

[3] 郭浩, 欧宗瑛. 基于 Gabor 滤波的指纹增强方法研究 [J]. 仪器仪表学报, 2003, 24(0z2):384-386.

[4] 楚亚蕴, 詹小四, 孙兆才, 等. 一种结合方向信息的指纹图像二值化算法 [J]. 中国图象图形学报, 2006, 11(006):855-860.

[5] Guang-Min L , Xue-Jun C . Improvement of OPTA algorithm and its application in fingerprint images thinning[J]. Computer Engineering and Design, 2006.

[6] 张晓康. 基于全局信息的指纹配准及在匹配中的应用 [D]. 2015.

[7] Ravi J , , Raja K B , R, V K . Fingerprint Recognition Using Minutia Score Matching[J]. International Journal of Engineering Science Technology, 2009, 1(2).

[8] 陈晖, 殷建平, 祝恩, 等. 一种基于细节点局部描述子的指纹图像匹配方法 [J]. 计算机工程与科学, 2010, 32(1):87-91.

[9] 陶刚. 基于结构特征的指纹识别系统的匹配算法研究 [D]. 华南师范大学, 2002.

[10] Chen C H , Chen C Y , Hsu T M . Singular Points Detection in Fingerprints Based on Poincare Index and Local Binary Patterns[J]. Journal of Imaging ence and Technology, 2019, 63(3):030401.1-030401.7.

# 附录 A 结果

指纹一：44.283 -1.510 0.748 1 4 46.872 -1.365 0.748 1 5 51.740 -1.236 0.805 1 6 19.026 -1.901 -0.459 1 0 13.928 -1.586 -0.286 1 1 28.792 -0.737 -0.435 1 2 65.521 -0.661 -0.616 1 6 71.868 0.818 -1.634 1 5 81.988 0.664 -2.331 1 6 34.132 -1.865 -0.358 2 0 96.773 -0.143 -0.692 2 12 39.051 -0.615 -0.499 2 3 51.225 0.513 0.760 2 4

指纹二：60.803 2.156 4.616 1 6 69.857 1.935 4.616 1 7 56.036 1.487 4.616 1 7 17.117 1.405 1.474 1 2 32.140 1.429 4.616 1 4 11.314 0.737 1.522 1 1 11.662 2.553 1.522 1 1 66.731 2.040 2.523 1 6 69.029 -0.019 4.613 1 6 166.018 0.327 1.932 2 10 70.937 2.123 1.474 2 2 140.071 1.423 2.195 2 1 112.969 1.201 4.616 2 4 23.022 1.767 2.496 2 9 118.152 1.840 2.499 2 7 40.200 2.317 2.606 2 7 41.110 2.456 2.709 2 8 98.955 2.764 2.731 2 8

指纹三：44.721 -1.731 -2.098 1 0 37.483 0.143 -0.848 1 3 28.018 -1.587 -0.544 1 0 31.016 -0.987 -0.553 1 0 79.101 -0.675 -0.848 1 5 11.180 -0.804 -0.694 1 0 64.382 -0.515 -0.693 1 2 14.318 0.514 -0.653 1 1 67.720 -0.908 -0.934 1 7 79.712 -0.970 -1.038 1 7 41.012 -1.409 -0.660 1 4 42.012 0.923 -0.480 1 4 72.422 -0.005 -0.438 1 8 53.600 0.307 -0.492 1 8 58.310 0.406 -0.492 1 9 94.069 -1.387 -0.668 1 11 90.907 0.255 -0.432 1 15 108.632 0.896 -1.396 2 2 39.051 -2.084 -0.721 2 1 27.166 0.161 -0.193 2 3

指纹四：81.633 0.177 2.060 1 6 90.802 0.273 2.272 1 8 47.927 0.310 1.548 1 5 47.539 0.462 2.075 1 6 19.416 0.115 1.006 1 2 42.190 0.622 1.411 1 4 40.311 0.309 0.452 1 4 98.489 0.943 2.405 1 11 88.119 0.233 -0.136 1 10 88.193 1.227 2.170 1 13 98.595 1.424 1.976 1 32 66.573 1.717 0.670 2 8 80.808 0.002 2.060 2 4 61.033 0.107 1.929 2 4 23.324 -0.313 0.780 2 3 124.310 1.003 0.520 2 7 149.967 0.107 0.628 2 1 53.160 1.677 0.677 2 1 12.207 0.968 1.438 2 4 12.207 0.209 -0.128 2 8

指纹五：76.381 0.398 -0.654 1 8 5.831 0.662 -0.369 1 1 9.849 -1.521 -0.189 1 1 66.068 0.513 0.158 1 6 82.680 0.641 0.307 1 9 34.986 0.662 -0.506 2 4 30.463 0.036 -0.553 2 3 194.258 -0.539 -0.017 2 11 58.856 -1.381 -0.189 2 1 133.060 -0.700 -0.757 2 6 119.017 -0.678 -0.909 2 7 18.868 -1.323 -0.211 2 2 67.676 0.692 -0.016 2 3 78.746 -0.030 -0.071 2 8 29.411 0.355 0.158 2 5 28.653 -1.097 -0.620 2 11 93.301 0.439 0.292 2 9

指纹六：76.381 0.398 -0.654 1 8 5.831 0.662 -0.369 1 1 9.849 -1.521 -0.189 1 1 66.068 0.513 0.158 1 6 82.680 0.641 0.307 1 9 34.986 0.662 -0.506 2 4 30.463 0.036 -0.553 2 3 194.258 -0.539 -0.017 2 11 58.856 -1.381 -0.189 2 1 133.060 -0.700 -0.757 2 6 119.017 -0.678 -0.909 2 7 18.868 -1.323 -0.211 2 2 67.676 0.692 -0.016 2 3 78.746 -0.030 -0.071 2 8 29.411 0.355 0.158 2 5 28.653 -1.097 -0.620 2 11 93.301 0.439 0.292 2 9

指纹七：95.603 -2.719 -4.295 1 11 94.372 -0.121 -1.355 1 8 94.578 -0.198 -1.355 1 8 81.320 -2.200 -1.358 1 8 59.506 -0.287 -1.347 1 3 36.346 -2.857 -4.003 1 3 34.928 -0.084 -1.398 1 2 21.095 0.052 -1.500 1 1 15.232 -0.258 -1.500 1 0 24.083 -1.341 -4.338 1 1 11.402 -1.158 -1.424 1 0 27.203 -2.053 -1.308 1 2 148.000 -0.530 -1.277 2 5 179.212 -2.120 -1.440 2 6 161.966 -0.657 -1.349 2 0 108.227 -2.580 -4.102 2 2 141.227 -1.054 -1.277 2 2 155.039 -1.244 -1.321 2 1 195.921 -1.341 -2.205 2 4 100.439 -1.690 -1.277 2 2 84.646 -0.258 -1.562 2 0 63.891 -1.109 -2.608 2 3 193.396 -1.728 -1.277 2 5 118.849 -0.984 -2.650 2 3 38.897 -2.183 -1.277 2 6 27.313 0.013 -1.739 2 3 52.555 -0.520 -2.630 2 4 22.361 -0.734 -2.764 2 5 120.037

-0.180 -1.918 2 5

指纹八：78.262 -0.133 -3.702 1 7 57.245 -0.008 -3.840 1 5 55.902 -0.133 -3.795 1 5 51.313 -0.247 -3.795 1 5 24.331 0.165 -1.003 1 0 28.302 -2.253 -1.033 1 2 32.202 -0.791 -3.810 1 4 68.964 -0.946 -2.058 1 7 55.154 -0.850 -1.900 1 5 62.626 -0.898 -2.058 1 6 93.392 -1.014 -2.424 1 9 54.562 -0.520 -2.049 1 5 60.141 -0.420 -2.327 1 6 152.411 -2.515 -3.851 2 8 129.016 0.285 -3.811 2 4 139.176 -2.134 -4.041 2 7 75.286 -2.474 -0.914 2 3 172.540 -1.122 -2.058 2 6 66.068 -2.623 -2.187 2 5 71.840 -0.133 -2.204 2 5

指纹九：56.613 -1.481 -0.516 1 16 58.822 -1.149 -0.460 1 17 81.216 -0.360 -1.538 1 11 28.636 -0.965 -0.379 1 3 7.071 0.252 -0.533 1 1 28.018 -0.569 -0.327 1 2 20.100 -0.433 -0.327 1 1 57.315 -0.428 -0.389 1 4 24.166 -0.960 -0.782 1 2 80.324 -0.203 -0.528 1 6 65.947 0.177 -0.227 1 12 71.063 0.996 -0.215 1 4 74.673 0.903 -0.215 1 4 63.063 -1.011 -0.589 2 14 38.897 -1.300 -0.379 2 9 32.573 -0.845 -1.175 2 3 67.417 -0.897 -1.775 2 5 103.392 0.444 -0.497 2 14

指纹十：58.523 2.196 0.425 1 5 57.706 2.058 0.425 1 5 9.220 2.327 0.974 1 0 10.296 -0.089 1.330 1 0 63.159 1.381 1.594 1 3 85.563 0.181 3.520 1 10 164.454 -0.503 1.055 2 6 110.386 1.804 3.520 2 10 105.380 1.749 3.520 2 6 75.326 -0.242 1.387 2 4 96.260 0.463 1.064 2 9 112.058 1.277 3.520 2 7 67.179 2.274 0.523 2 1 49.041 1.438 0.423 2 2 94.021 1.132 3.520 2 7 80.654 0.833 1.351 2 1 18.682 0.266 0.467 2 1 29.069 1.170 1.717 2 5 69.871 0.445 3.520 2 6 35.355 2.036 1.491 2 3 106.170 1.634 1.761 2 4 18.439 1.795 1.840 2 5

指纹十一：59.464 -0.251 1.314 1 6 27.514 1.310 0.850 1 2 23.022 1.020 0.783 1 2 2.236 -0.130 0.977 1 0 32.388 0.822 1.100 1 4 15.000 0.333 0.783 1 2 57.585 0.622 0.761 1 5 82.462 1.222 0.777 1 6 64.078 1.378 0.818 1 5 75.743 1.809 1.141 1 7 69.921 2.258 1.187 1 7 98.671 -0.285 1.210 1 10 73.430 1.489 0.430 2 6 79.831 1.191 0.383 2 6 33.734 1.186 0.850 2 2 99.464 1.200 0.615 2 7 62.642 -0.302 1.223 2 6 82.420 1.926 1.235 2 7

指纹十二：71.169 -2.842 -1.571 1 4 14.036 -1.642 -1.571 1 1 36.056 -1.626 -1.571 1 3 18.439 -2.279 -1.571 1 1 32.757 -2.116 -1.571 1 2 123.483 -2.248 -1.571 2 3 138.293 -1.636 -1.571 2 2 78.243 -1.435 -1.571 2 2 46.098 -0.636 -1.571 2 4 73.682 -2.921 -1.571 2 6

指纹十三：74.277 0.228 1.562 1 5 72.367 1.236 0.578 1 6 41.110 0.720 1.096 1 1 4.472 0.578 1.067 1 0 26.401 0.391 1.196 1 3 19.925 -0.223 1.111 1 2 69.584 -0.399 1.148 2 4 6.000 1.042 1.042 2 0 62.032 1.074 1.032 2 2 47.170 0.957 1.155 2 4 103.121 0.490 1.143 2 1 15.264 0.454 1.304 2 3 36.056 0.544 1.257 2 4 52.355 2.240 1.097 2 4

指纹十四：75.286 2.450 4.712 1 9 62.610 2.678 4.712 1 7 50.596 2.820 4.712 1 6 54.489 0.746 1.866 1 4 93.434 1.474 4.422 1 3 34.928 1.983 4.638 1 3 44.385 1.964 4.638 1 3 58.822 1.260 4.712 1 7 68.964 1.865 4.488 1 4 50.537 2.043 4.638 1 2 60.415 2.071 4.547 1 5 90.554 2.609 4.700 1 9 226.080 0.686 4.712 2 3 28.425 2.290 1.653 2 6 138.101 2.815 4.712 2 6 85.041 2.774 4.712 2 9

指纹十五：68.797 -0.936 0.338 1 1 64.846 0.364 -0.540 1 6 5.000 0.067 0.371 1 0 35.355 -0.075 -0.360 1 5 92.195 0.001 -0.540 1 12 9.220 0.929 0.371 1 0 39.825 -0.612 -0.348 1 5 51.313 -0.927 0.024 1 6 111.987 -1.282 0.149 2 4 86.122 0.000 -0.074 2 5 78.000 -1.332 -0.007 2 3 117.924 0.434 0.067 2 1 70.036 -0.080 0.730 2 1 143.753 0.264 1.344 2 2 13.928 -0.494 -0.430 2 5 34.366 1.456 -1.427 2 4 228.055 -0.989 -0.554 2 7 50.990 -1.101 -0.716 2 9

指纹十六：18.974 -0.360 0.880 1 2 89.140 -0.626 1.227 1 10 98.509 -0.580 1.227 1 11 126.194 -0.074 0.960 2 8 96.260 -0.284 0.780 2 5 82.462 -0.134 0.882 2 4 84.380 1.660 0.265 2 10 97.591 0.141 0.932 2 7 94.112 1.269 1.038 2 4 142.678 0.663 0.871 2 4 48.466 1.167 0.977 2 3 62.586 0.834 0.634 2 6 36.401 1.237 0.876 2 1 91.137 0.793 0.253 2 9 11.705 -0.012 1.008 2 3 83.385 1.468 0.774 2 5 128.316 -0.496 1.282 2 10

# 附录 B 代码

## B.1 Gabor matlab 源程序

```
%------------------------------------------------------------------------
%gabor_kernel
%creates a 2D gabor convolution mask
%------------------------------------------------------------------------
function [gr,gi] = gabor_kernel(dx,dy,f,theta)
   dx    = round(dx);
   dy    = round(dy);
   [x,y] = meshgrid(-3*dx:3*dx,-3*dy:3*dy);
   xp    = x*cos(theta)+y*sin(theta);
   yp    = -x*sin(theta)+y*cos(theta);
   gr     = exp(-xp.^2/dx.^2-yp.^2/dy.^2).*cos(2*pi/f*xp);
   gi     = exp(-xp.^2/dx.^2-yp.^2/dy.^2).*sin(2*pi/f*xp);
%end function gabor_kernel
```

## B.2 problem1.m matlab 源程序

```
  function yi=prepro_1(img,msk);
N=16;
msk=pad_image(msk,N);
img =  double(img);
y_img=img;
img = pad_image(img,N);
[ht,wt]=size(img);
   nimg   =   normalize_image(img,0,100);
   %---------------------------------------
   %orientation image
   %---------------------------------------
   oimg          =   blk_orientation_image(img,N);
   %---------------------------------------
   %smoothen orientation image
   %---------------------------------------
```

```
    oimg          =   smoothen_orientation_field(oimg);
%求指纹频率场
    [x,y]         =   meshgrid(-8:7,-16:15);
    [blkht,blkwt] =   size(oimg);
    [ht,wt]       =   size(img);
    yidx = 1; %index of the row block
    for i = 0:blkht-1
        row    = (i*N+N/2);%+N for the pad
        xidx   = 1; %index of the col block
        for j = 0:blkwt-1
            col = (j*N+N/2);
            %row,col indicate the index of the center pixel
            th  = oimg(yidx,xidx);
            u = x*cos(th)-y*sin(th);
            v = x*sin(th)+y*cos(th);
            u          =   round(u+col); u(u<1) = 1; u(u>wt) = wt;
            v          =   round(v+row); v(v<1) = 1; v(v>ht) = ht;
            %find oriented block
            idx        =   sub2ind(size(img),v,u);
            blk        =   img(idx);
            blk        =   reshape(blk,[32,16]);
            %find x signature
            xsig       =   sum(blk,2);
            f(yidx,xidx) = find_peak_distance(xsig);
            xidx = xidx +1;
        end;
        yidx = yidx +1;
    end;
    fimg=filter_frequency_image(f);
    y = do_gabor_filtering(img,oimg,fimg); %滤波增强图像gobor
   yi=imscale(y);
```

### B.3 problem2.m 源程序

```
   clear, clc;
[endpoints5, bifurcations5] = get_code(10);
[endpoints6, bifurcations6] = get_code(11);
for i = 1:size(endpoints5,1)
    feature1(i,:)= [endpoints5(i,7), endpoints5(i,5), endpoints5(i,6), 1,
        endpoints5(i,3)];
end
for i = size(endpoints5,1)+1:size(endpoints5,1)+size(bifurcations5,1)
```

```
    feature1(i,:)= [bifurcations5(i-size(endpoints5,1),7),
        bifurcations5(i-size(endpoints5,1),5),
        bifurcations5(i-size(endpoints5,1),6), 2,
        bifurcations5(i-size(endpoints5,1),3)];
end

for i = 1:size(endpoints6,1)
    feature2(i,:)= [endpoints6(i,7), endpoints6(i,5), endpoints6(i,6), 1,
        endpoints6(i,3)];
end
for i = size(endpoints6,1)+1:size(endpoints6,1)+size(bifurcations6,1)
    feature2(i,:)= [bifurcations6(i-size(endpoints6,1),7),
        bifurcations6(i-size(endpoints6,1),5),
        bifurcations6(i-size(endpoints6,1),6), 2,
        bifurcations6(i-size(endpoints6,1),3)];
end
for i = 1:size(feature1,2)
    feature1(:,i) =
        (feature1(:,i)-min(feature1(:,i)))/(max(feature1(:,i))-min(feature1(:,i)));
    feature2(:,i) =
        (feature2(:,i)-min(feature2(:,i)))/(max(feature2(:,i))-min(feature2(:,i)));
end
d = zeros(size(feature1,1),size(feature2,1));
for i = 1:size(feature1,1)
    for j = 1:size(feature2,1)
        d(i,j)=sqrt((feature1(i,1)-feature2(j,1))^2+(feature1(i,2)-feature2(j,2))^2+(feature1
    end
end
fid = fopen('d.txt','w');
for i = 1:size(d,1)
    for j = 1:size(d,2)
        fprintf(fid, '%.4f ', d(i,j));
    end
    fprintf(fid, '\n');
end
fclose(fid);
```

### B.4 smoothen orientation field–matlab 源程序

```
  function oimg = smoothen_orientation_field(oimg)
   g   =   cos(2*oimg)+i*sin(2*oimg);
   g   =   imfilter(g,fspecial('gaussian',5));
```

```
    oimg=  0.5*angle(g);
    save('oimg1.mat','oimg');
%end function smoothen_orientation_field
%blk_frequency_image
```

### B.5 get code–matlab 源程序

```
function [endpoints, bifurcations]=get_code(ord)
[endpoint_x, endpoint_y,bifurcation_x,bifurcation_y, endpoint_line_num,
    bifurcation_line_num, corepoint_y, corepoint_x,oimg] = solve_pro(ord);
%% 筛选特征点
R = 100;
% 对端点进行筛选
endpoints = []; % 第一列为坐标，第二列为坐标，第三列为穿过脊的数量，第四列为方
    向场，第五列为极坐标系下的角度xy
            % 第六列为方向场角度差值，第列为距离，第列为 78
bifurcations = [];
for i = 1:length(endpoint_x)
   if sqrt((corepoint_x-endpoint_x(i))^2+(corepoint_y-endpoint_y(i))^2) <= R
      endpoints(end+1,1)=endpoint_x(i);
      endpoints(end,2) = endpoint_y(i);
      endpoints(end,3) = endpoint_line_num(i);
   end
end
for i = 1:length(bifurcation_x)
   if
       sqrt((corepoint_x-bifurcation_x(i))^2+(corepoint_y-bifurcation_y(i))^2)
       <= R
      bifurcations(end+1,1)=bifurcation_x(i);
      bifurcations(end,2) = bifurcation_y(i);
      bifurcations(end,3) = bifurcation_line_num(i);
   end
end

%% 计算方向场角度差
% 中心点所在角度场
core_angle = oimg(floor(corepoint_y/16), floor(corepoint_x/16));
% 计算端点
for i = 1:size(endpoints,1)
   endpoints(i,4) = oimg(floor(endpoints(i,2)/16),floor(endpoints(i,1)/16));
   endpoints(i,4) = endpoints(i,4)-core_angle;
   if endpoints(i,1) == corepoint_x
```

```
      endpoints(i,5) = 0;
   else
      endpoints(i,5) =
          atan((endpoints(i,2)-corepoint_y)/(endpoints(i,1)-corepoint_x))-core_angle;
   end
   endpoints(i,6) = endpoints(i,4)-core_angle;
   endpoints(i,7) =
       sqrt((corepoint_x-endpoints(i,1))^2+(corepoint_y-endpoints(i,2))^2);
end
%计算交叉点
for i = 1:size(bifurcations,1)
   bifurcations(i,4) =
       oimg(floor(bifurcations(i,2)/16),floor(bifurcations(i,1)/16));
   bifurcations(i,4) = bifurcations(i,4)-core_angle;
   if bifurcations(i,1) == corepoint_x
      bifurcations(i,5) = 0;
   else
      bifurcations(i,5) =
          atan((bifurcations(i,2)-corepoint_y)/(bifurcations(i,1)-corepoint_x))-core_angle
   end
   bifurcations(i,6) = bifurcations(i,4)-core_angle;
   bifurcations(i,7) =
       sqrt((corepoint_x-bifurcation_x(i))^2+(corepoint_y-bifurcation_y(i))^2);
end
```

### B.6 get picture–matlab 源程序

```
clear, clc;
for i = 7:16
[img] = solve_pro(i);
figure,imshow(img);
saveas(gcf, [num2str(i),' '], 'png')
end
```

### B.7 问题二模型求解 –lingo 源程序

```
model:
sets:
row1/1..22/;
row2/1..18/;
col/1..5/;
```

```
dist(row1,row2):d;
mat(row1, row2):x,z,p;
endsets
data:
d = 1.3979 0.6704 0.6900 1.2423 0.9313 0.8982 0.7611 0.7579 0.6387 0.9542
    0.9641 1.5458 1.1176 1.1739 1.2166 1.3047 1.6811 1.4630
1.3668 0.6478 0.6594 1.2065 0.9052 0.8593 0.7255 0.7397 0.6202 0.9513
    0.9697 1.5187 1.1095 1.1616 1.2025 1.2946 1.6549 1.4629
1.5304 0.5911 0.6186 1.0399 0.9599 0.8280 1.0170 1.1245 0.9255 1.2235
    1.2018 1.8640 1.4151 1.4839 1.1938 1.6327 1.8087 1.6524
1.1020 0.6177 0.4886 0.3541 0.7522 0.2909 0.7907 1.1172 0.9677 1.3354
    1.4210 1.5310 1.5005 1.5043 1.1720 1.6347 1.4598 1.7523
0.9757 0.1851 0.2298 0.7979 0.4654 0.4836 0.4277 0.5702 0.3624 0.7363
    0.7928 1.2649 1.1605 1.2012 1.0164 1.2683 1.3725 1.3107
0.4674 1.0452 1.0573 1.1846 0.6962 1.0465 0.7892 0.8539 0.8522 0.7309
    0.8517 0.5637 1.5263 1.5397 1.4254 1.4251 1.1156 1.2662
1.3428 1.4809 1.4478 1.5977 1.4246 1.4238 1.1691 1.1971 1.2816 1.4567
    1.5928 1.2760 0.7650 0.6503 1.0216 0.5970 0.7948 1.1421
1.3403 1.4408 1.4987 1.7650 1.2926 1.5992 1.3474 1.2536 1.2486 1.0656
    1.0699 1.3349 1.0401 1.1114 1.0248 0.9094 0.9138 0.3608
1.2661 1.2510 1.3144 1.5511 1.1471 1.4224 1.2467 1.1898 1.1460 1.0315
    1.0374 1.3907 0.9612 1.0378 0.7359 0.8730 0.8019 0.2547
1.2435 1.1803 1.1310 1.2036 1.1596 1.0795 1.0545 1.2104 1.1819 1.4102
    1.5140 1.4215 0.7558 0.7189 0.5710 0.8174 0.6609 1.1044
1.3501 1.3231 1.3003 1.5444 1.2805 1.3163 1.0955 1.1274 1.1544 1.2962
    1.3866 1.3311 0.5219 0.4978 0.8309 0.5431 0.8333 0.9439
1.1773 1.2885 1.3375 1.5413 1.1414 1.4150 1.2136 1.1707 1.1511 1.0400
    1.0759 1.2641 0.9601 1.0151 0.7842 0.8242 0.6455 0.3030
1.7768 1.1692 1.1872 1.5336 1.3902 1.3282 1.3126 1.3250 1.2345 1.4636
    1.4680 1.9685 0.6771 0.7680 0.6269 0.9775 1.4257 1.1643
1.6473 1.1203 1.1024 1.3768 1.3028 1.1798 1.2076 1.2924 1.2056 1.4770
    1.5124 1.8523 0.6237 0.6810 0.5171 0.9353 1.2565 1.2019
1.1502 1.2581 1.2998 1.4766 1.1035 1.3623 1.2000 1.1921 1.1566 1.0674
    1.1042 1.2757 0.9805 1.0361 0.7400 0.8772 0.5993 0.4126
1.4003 1.0563 1.0438 1.2247 1.1554 1.0886 1.0934 1.1889 1.1194 1.3552
    1.4184 1.6295 0.6620 0.6830 0.2852 0.8168 0.9287 1.0151
1.6012 1.1899 1.1253 1.2020 1.3237 1.0901 1.2570 1.4512 1.3580 1.6700
    1.7364 1.8738 0.9409 0.9438 0.6426 1.1817 1.1949 1.4539
1.3738 1.0564 1.0503 1.2607 1.0911 1.0990 1.1171 1.2232 1.1247 1.2830
    1.3147 1.5998 0.7226 0.7962 0.3682 0.9105 0.9155 0.9225
1.0670 1.2297 1.2447 1.3098 1.0536 1.2490 1.1811 1.2607 1.2070 1.1907
    1.2506 1.2813 1.0731 1.1080 0.6879 1.0040 0.4183 0.7015
```

```
  1.5702 1.0557 1.0790 1.3760 1.1921 1.1999 1.2215 1.2723 1.1541 1.3156
     1.3107 1.8046 0.7306 0.8390 0.3984 0.9784 1.1882 0.9576
  1.3800 1.0925 1.1250 1.4373 1.1460 1.2411 1.0813 1.0516 1.0105 1.1216
     1.1580 1.5044 0.4477 0.5351 0.4068 0.5411 0.9123 0.6138
  1.4871 1.0795 1.0963 1.3568 1.1444 1.1890 1.2108 1.2824 1.1649 1.2855
     1.2827 1.7186 0.8008 0.9041 0.4639 1.0094 1.0879 0.9157 ;
  enddata
  max = M/22;
  L = 1.0;
  @for(row2(j):@sum(row1(i):x(i,j))<1);
  @for(row1(i):@sum(row2(j):x(i,j))<1);
  @for(row1(i):@for(row2(j):(1-z(i,j))>x(i,j)*d(i,j)-L));
  @for(row1(i):@for(row2(j):p(i,j)=x(i,j)*z(i,j)));
  @for(row1(i):@for(row2(j):p(i,j)<x(i,j)));
  @for(row1(i):@for(row2(j):p(i,j)<z(i,j)));
  @for(row1(i):@for(row2(j):x(i,j)+z(i,j)-1<p(i,j)));
  M = @sum(row1(i):@sum(row2(j):z(i,j)*x(i,j)));
  @for(mat:@bin(x));
  @for(mat:@bin(z));
```

### B.8 get main–matlab 源程序

```
  clear,clc;
  [endpoints, bifurcations] = get_code(1);
  fid = fopen('result.txt','wb');
  for i = 1:size(endpoints,1)
     fprintf(fid, '%.3f %.3f %.3f %d %d\n', endpoints(i,7), endpoints(i,5),
        endpoints(i,6), 1, endpoints(i,3));
  end
  for i = 1:size(bifurcations,1)
     fprintf(fid, '%.3f %.3f %.3f %d %d\n', bifurcations(i,7),
        bifurcations(i,5), bifurcations(i,6), 2, bifurcations(i,3));
  end
  fclose(fid);isting}
```

### B.9 细化 –matlab 源程序

```
  function c=thin_img(iseg, msk)

c=bwmorph(iseg,'thin','inf'); %
fun=@minutie;
```

```
c = nlfilter(c,[3 3],fun);

i_thin=c(5:364,1:256);
msk=msk(5:364,1:256); % -----------ó--
i_thin(i_thin>0)=1;

% imshow(c);
% title细(' 化');
```