

# AI HW1: 二值图像降噪

组号:X 组员:朱俊丰 卓招进 张泽昕 朱新强

## 问题描述

对于一幅真彩图片，我们先通过一个恰当的方法将其转换为二值图像，之后并对其随机地加上10%的噪点，生成一幅含有噪点的图像。而我们的目的便是寻找一个恰当的方法和算法，对其进行去噪处理，生成一幅恢复后的图像来尽可能的逼近原来的加噪前的图像。为了评定算法的优劣，我们将恢复后的图像与原图像进行每个位置的像素点的比较，计算其恢复率。

## 模型

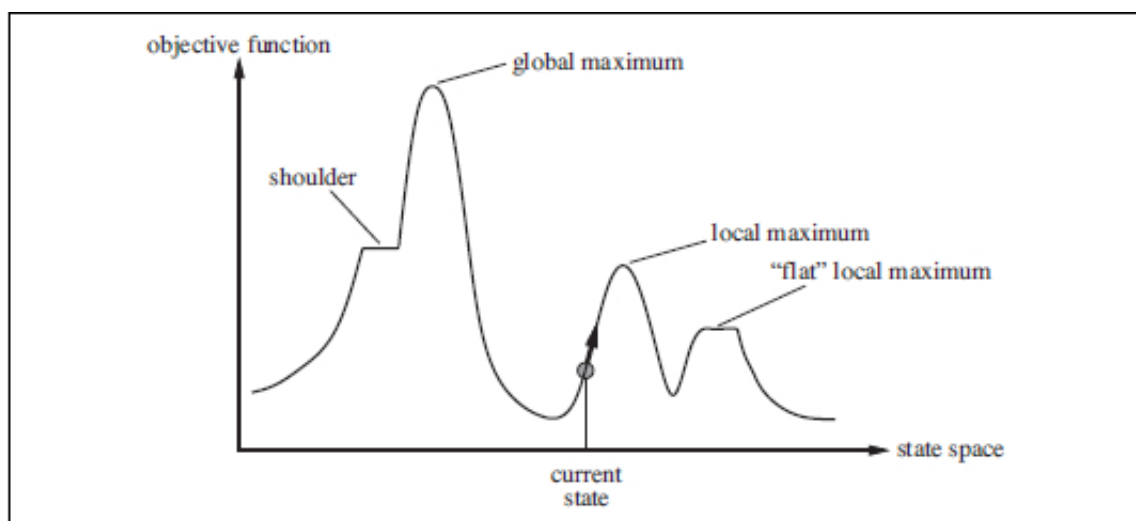
### 二值图像模型

对于一幅N位位图（1、4、8、16、24及32位等），我们为了将其二值化时尽量保存其边界信息，于是对于每个像素的某个通道的值，当它大于127时，我们都统一将其扩大为255，否则则降低为0。

### 加噪模型

对于一幅二值图像，我们先根据其像素点的总个数（图像长度\*图像宽度，记其为S）计算出噪点个数，记其为M。通过随机函数产生M个随机数A，其中 $A_i \leq S (i=1,2,3...M)$ 。对于每个由 $A_i$ 确定的像素点，我们对其进行模糊操作（对于二值图像，便是取反操作）。

### 爬山法模型



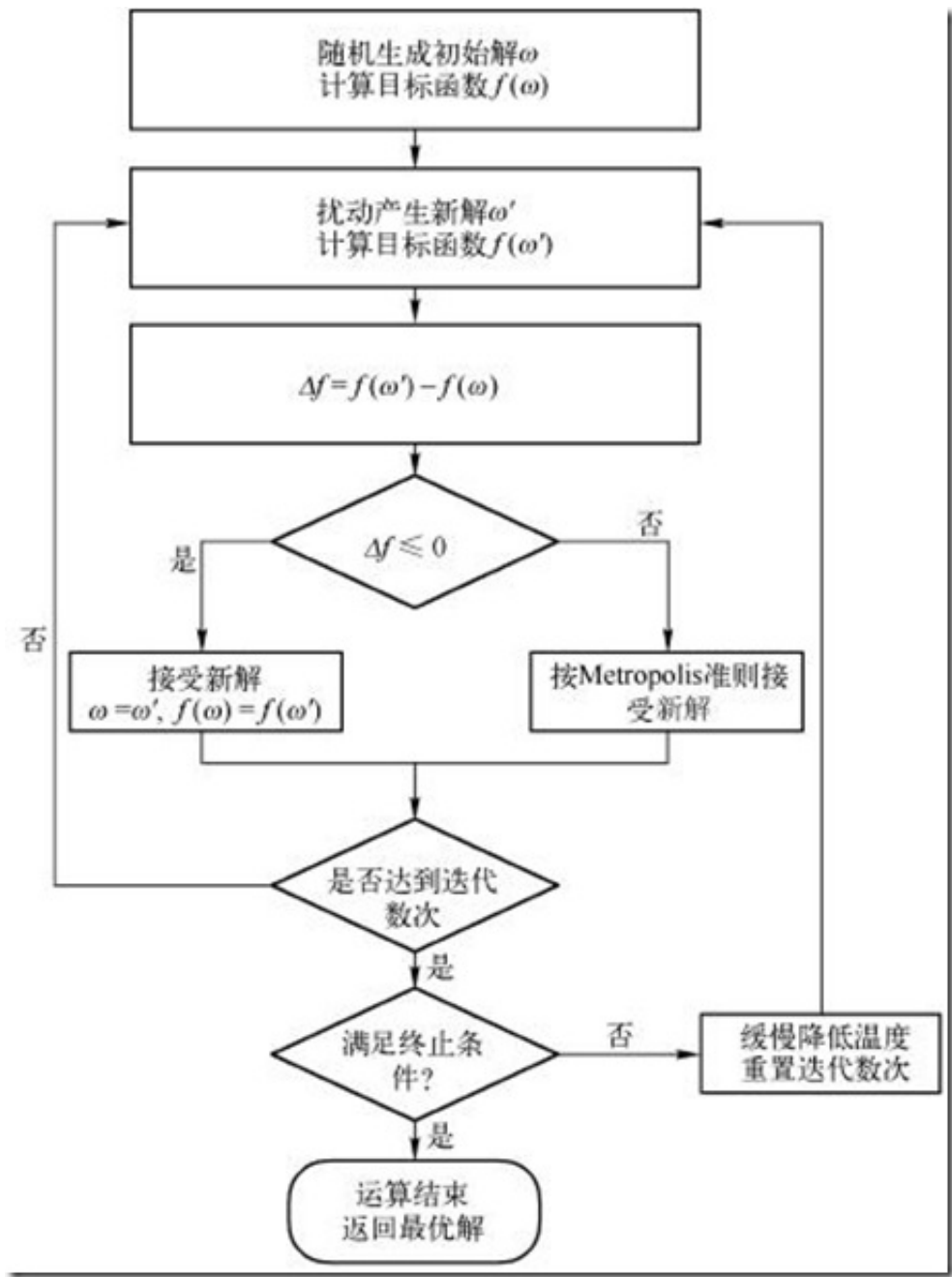
爬山法是一个搜索算法，根据一个elevation（评价函数），评价函数的意义（决定寻找的是最大值还是最小值），在状态空间中寻找一个最能满足要求的状态。爬山法每次move只会往前看一步，并不会生成一个搜索树，只保存当前的状态以及对应的评价函数的值。导致的后果就是爬山法往往只能找到一个局部最大值，其最大的缺点就是不能下山。如图所示，current state会逐渐走到local maximum上面而不是global maximum。

能量计算模型

对于一个点(x,y)，以及在这个点的像素值为R，分别计算其周围的若干个点的像素值，若其值为R（或者接近）的比例越大，则其能量越小。

模拟退火算法模型

模拟退火算法是在爬山法的基础上进行改良而得到的。爬山法是完完全全的贪心法，每次都鼠目寸光的选择一个当前最优解，因此只能搜索到局部的最优值。模拟退火其实也是一种贪心算法，但是它的搜索过程引入了随机因素。模拟退火算法以一定的概率来接受一个比当前解要差的解，因此有可能会跳出这个局部的最优解，达到全局的最优解。模拟退火算法的流程框图如下：



算法

二值化算法

对于一幅普通的彩色图像，我们对其所有像素点进行遍历扫描。对于每个像素点，我们选取它的Red通道值

之后再将修改后的像素值保存为图像。可以写出伪代码如下：

```
/// @describe binary the normal image to a binary image
///
/// @para inputImage the normal image
/// @return outputImage image after binaryzation
func binaryzation(inputImage)->outputImage
    let pointArray = arrayFromImage(inputImage)
    for i in pointArray do
        let r = getRChannel(i)
        if r > 127
            r = 255
        else
            r = 0
    return imageFromArray(pointArray)
```

## 加噪算法

对于一幅二值图像，我们随即选取出10%个像素点，然后对这些像素点进行取反操作：选取它的Red通道值，当Red为0时，将Red置为255;当Red为255时，将其置为0，之后再将修改后的像素值保存为图像。可以写出伪代码如下：

```
/// @describe add noise to a binary image
///
/// @para inputImage a binary image
/// @return noise image
func addNoise(inputImage) -> outputImage
    let pointArray = arrayFromImage(inputImage)
    let randomArray = random select 10% points from pointArray
    for i in randomArray do
        revert(i)
    return imageFromArray(pointArray)

/// @describe revert a point
///
/// @para point the input point
/// @return the reversed point
func revert(point) -> revervedPoint
    let r = getRedChannel(i)
    if r == 255
        r = 0
    else
        r = 255

    return redToPoint(r)
```

## 爬山法实现的降噪算法：

首先需要定义一个能量评价函数（能量越低越好的），可以根据先验知识知道：

- 一个点会跟周围的点具有相同的值，即图形一般都是有形状的，而不是以散点形式存在在图片上的。
- 由于噪点只有10%，大多数的点都是完好无损的，所以在将颜色反转前应当考虑是否真的是噪点，即每个像素点总是想保持原有状态的。

所以根据这些可以写出能量计算模型的伪代码如下：

```
/// @describe Caculate energy for input point
///
/// @para input The point input
/// @return the energy calculate from the 8 points near the input point
func energy(input) -> energy
    let E = 0
    let W1 = 3
    let neighborArray = select neighbor (W1*W1-1) points for input
    for point in neighborArray do
        if input == point
            increase(E)
    return -1*E
```

接下来就是扫描的工作，针对每一个像素点，把它设为黑和白，然后比较两种情况下的能量函数的取值，选取能够使能量变小的那个取值设为该像素点的值。伪代码如下：

```
/// @describe Denoise an noised image using climbing algorithm
///
/// @para inputImage The noised image
/// @return outputImage An image after denoise
func denoise(inputImage) -> outputImage
    let pointArray = arrayFromImage(inputImage)
    for i in pointArray do
        let E1 = energy(i)
        let Ri = revert(i)
        let E2 = energy(i)
        if E1 > E2
            i = Ri
    return imageFromArray(pointArray)
```

这就是普通的爬山法在降噪算法中的运用，我们将其命名为一级爬山法。

## 调整后的二级爬山法实现的降噪算法

我们在实验过程中发现，对于普通的爬山算法，我们去噪的效果并不好，仍然会有不少的噪点无法有效的消去。（见实验结果图1~4）

为此我们经过思考和讨论后认为，这是因为对于每一个点，我们考虑的点只有其周围的8个点，有时候并不能很好的将其判断为“噪点”，所以我们尝试将其考虑范围扩大。在经过多次试验比较后，我们改为考虑其周围 $5 \times 5 - 1$ 共24个点，根据24个点的值来决定这个点的取值，这样能最大程度的把我们增加的噪点消除。现在的能量计算伪代码如下：

```

/// @describe Caculate energy for input point
///
/// @para input The point input
/// @return the energy calculate from the 24 points near the input point
func energy(input) -> energy
    let E = 0
    let W2 = 5
    let neighborArray = select neighbor (W2*W2-1) points for input
    for point in neighborArray do
        if input == point
            increase(E)
    return -1*E

```

这就是训练了参数weight后的爬山法，我们将其命名为二级爬山法。

## 一级爬山法+二级爬山法组合并随机选取路径算法

我们在实验的时候也发现，经过改良后的二级爬山法虽然能消除大部分增加的噪点，但它却会很容易将原来图像里的孤立的点（如在图像边缘的点，见实验结果图5）。同时我们发现，这个现象在一级爬山法里并不会出现。于是我们经过讨论和思考后认为：这是因为对于一级爬山法，每一个点只考虑距离其最近的8个点，这8个点的距离相同，所以比重一样。而二级爬山法对于周围24个点，距离并不是都是一样的，所以我们认为应该是最近的8个点的权重应该高于次近的16个点的距离。

同时，我们还注意到因为顺序选取点的话，会导致递归影响的现象，即如果点1被认为是黑的，那么与它临近的点2被认为是黑的概率就会比较大，这又会导致点3被认为是黑的概率比较大.....以此类推。于是，我们每次选取的点都通过一个随机函数来获得，以尽量消除这种影响。现在的伪代码如下：

能量计算部分：

```

/// @describe Caculate energy for input point
///
/// @para input The point input
/// @return the energy calculate from the 24 points near the input point
func energy(input) -> energy
    let E = 0
    let W1 = 3
    let W2 = 5
    let neighborArray1 = select neighbor (W1*W1-1) points for input
    let neighborArray2 = select neighbor (W2*W2-1) points for input

    for point in neighborArray1 do
        if input == point
            increase(E)
            increase(E) //increase twice

    for point in neighborArray2 do
        if input == point
            increase(E) //increase once

    return -1*E

```

降噪部分:

```
/// @describe Denoise an noised image using climbing algorithm with random select a path
/// @inputImage The noised image
/// @return An image after denoise
func denoise(inputImage) -> outputImage
    let T = T0
    let control = a float number from 0 to 1
    let randomArray = randomArrayFromImage(inputImage)
    for i in randomArray do
        let E1 = energy(i)
        Ri = revert(i)
        let E2 = energy(Ri)
        if E1 <= E2
            i = Ri

    return imageFromArray(randomArray)
```

## 模拟退火实现的降噪算法:

经过改良后的这种算法可以说效果已经达到了近乎完美的地步。通过训练参数weight、weight1、weight2也让我们注意到了随机算法可以让整体的结果提升，于是我们考虑加入模拟退火算法来优化爬山法的不足。

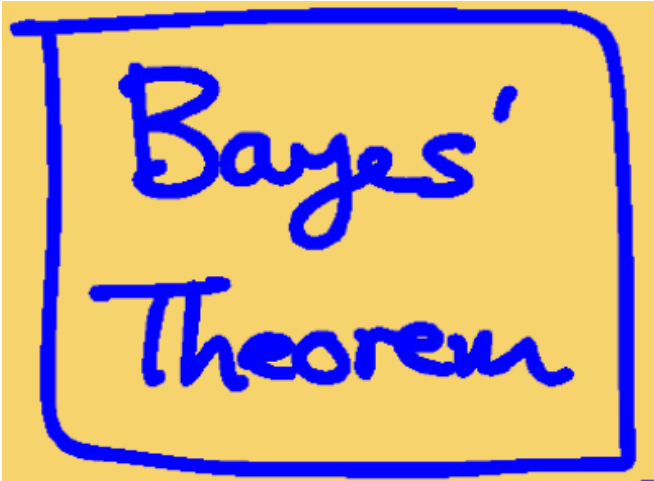
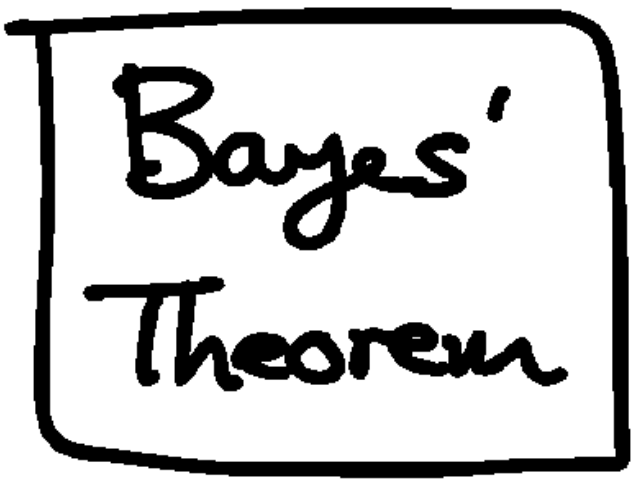
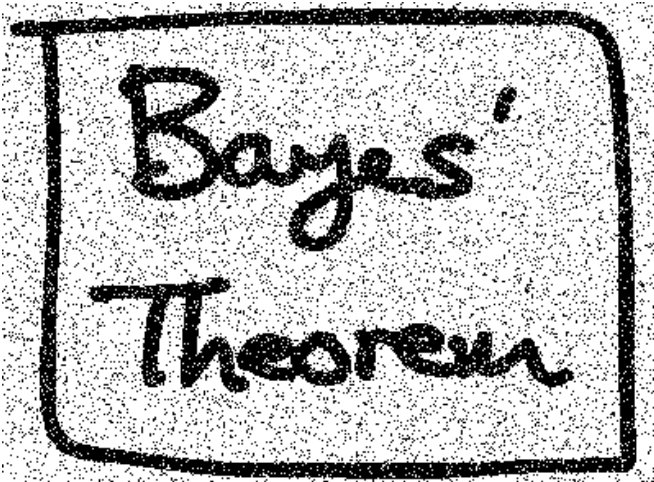
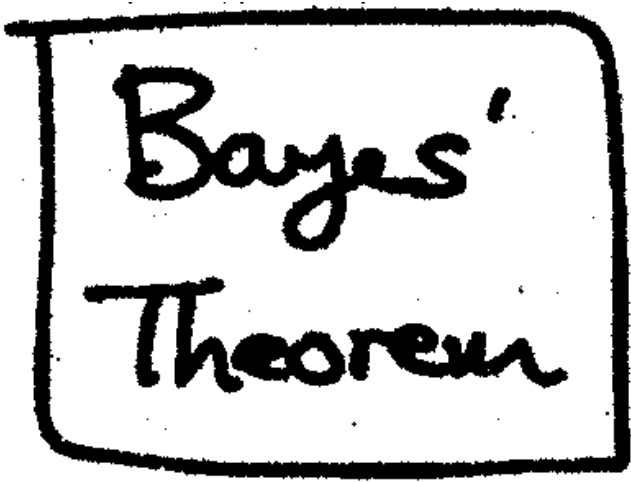
在退火之前我们有一个初始温度T，对于一个点，我们考虑其在变化之后的能量变化，若果其能量变小了，则100%接受改变，若能量变大了，则有一定的概率R接受改变。并且每次接受能量变大的改变，T的值和R的值都会按照一定的系数减小。写成伪代码如下：

```
/// @describe Denoise an noised image using simulate annealing algorithm
/// @inputImage The noised image
/// @return An image after denoise
func denoise(inputImage) -> outputImage
    let T = T0
    let control = a float number from 0 to 1
    let randomArray = randomArrayFromImage(inputImage)
    for i in randomArray do
        let E1 = energy(i)
        Ri = revert(i)
        let E2 = energy(Ri)

        if E1 <= E2
            i = Ri
        else
            let probability = calculate probability from T
            T = control * T
            if hit the probability
                i = Ri
    return imageFromArray(randomArray)
```

## 实验结果

一级爬山法

图1、实验原图	图2、二值化后的图像
	
图3、加噪后的图像	图4、一级爬山法的效果图
	

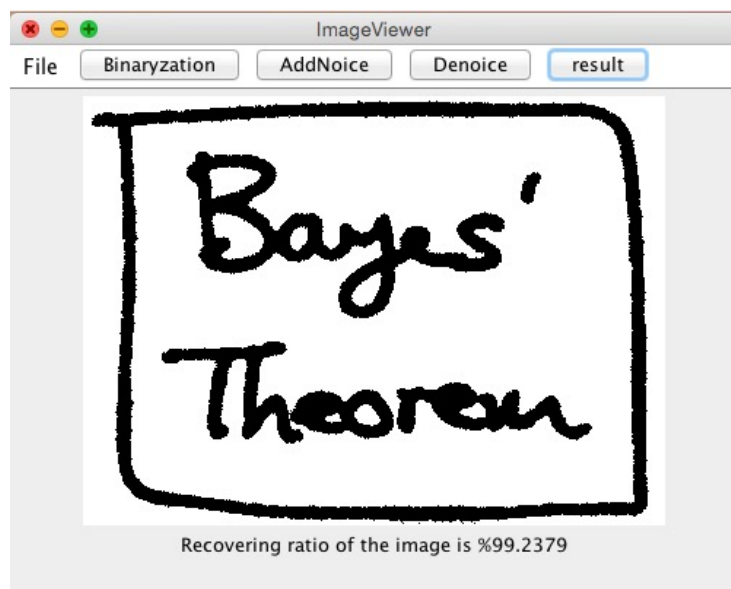
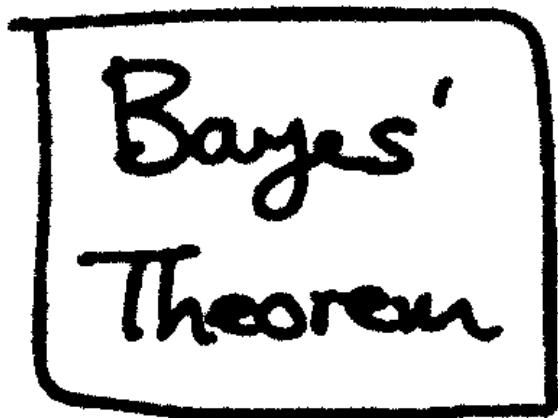
可以看到，一级爬山法很明显有一些噪点并不能有效的去除，经过100次的加噪、去噪之后，恢复的结果稳定在原图的99.2%左右。

二级爬山法



图5、二级爬山法的效果图

实验结果

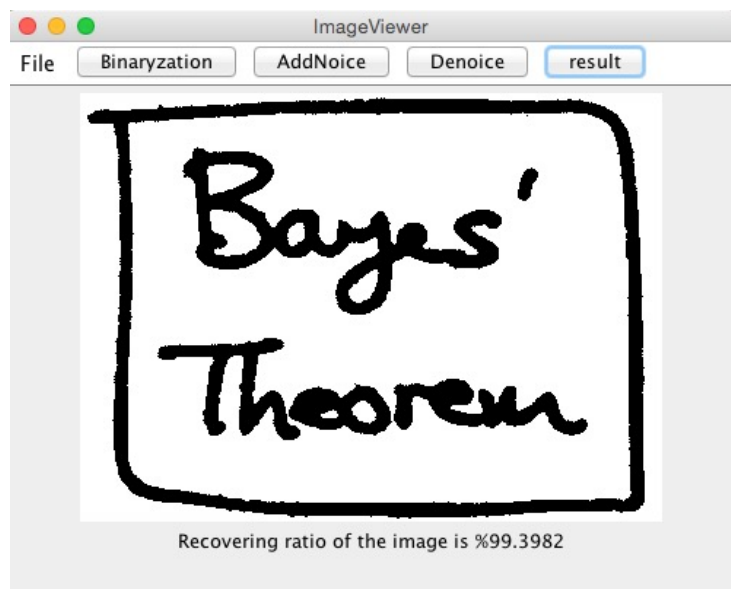
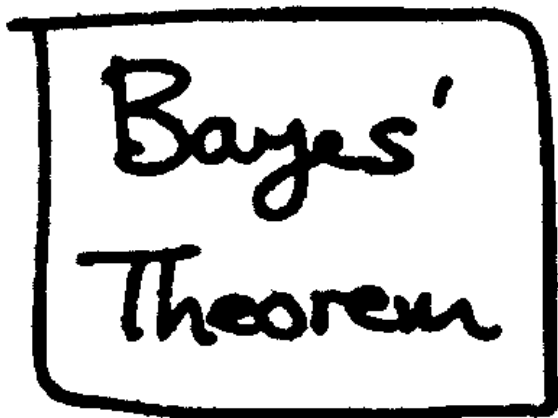


可以看到，二级爬山法能有效的去除之前增加的噪点，但在图像边缘的点会被当作噪点也去除了，所以恢复的结果跟一级爬山法接近，在原图的99.25%左右。

### 一级爬山法+二级爬山法+随即选取路径算法

图6、一级爬山法+二级爬山法+随即选取路径效果图

实验结果

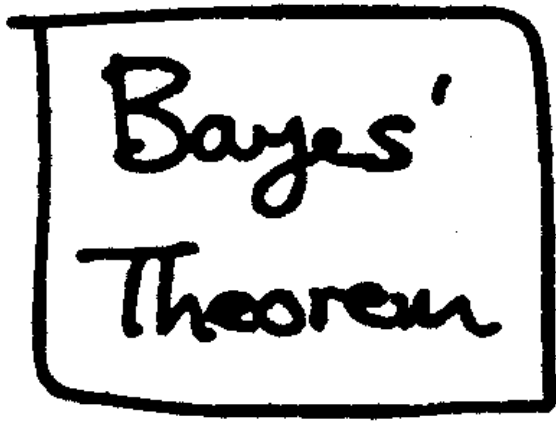


可以看到，经过进一步优化后的算法，实现的效果既能有效的去除噪点，又能更好地保留边缘信息，可以说效果是相当不错的，结果稳定在原图的99.4%左右。

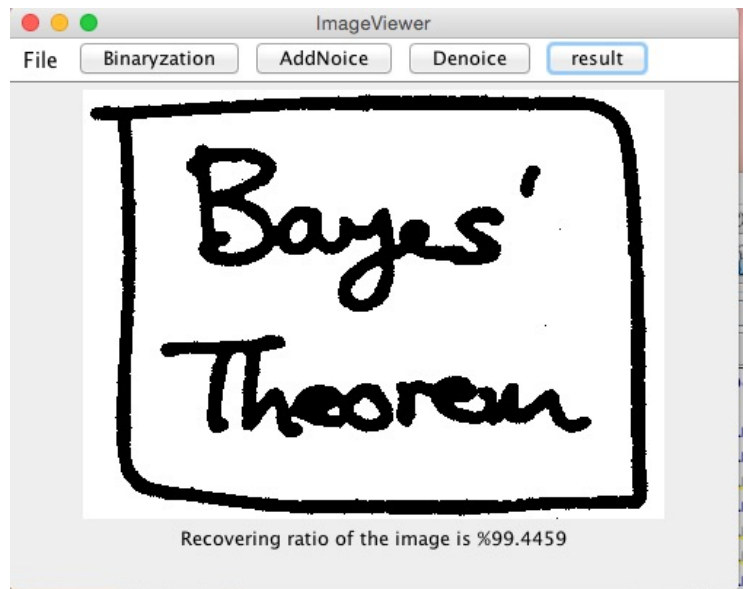
### 模拟退火算法



图7、模拟退火算法改良后的效果图



实验结果



使用了模拟退火算法改良后，效果进一步加强，大部分结果都在99.4%以上。