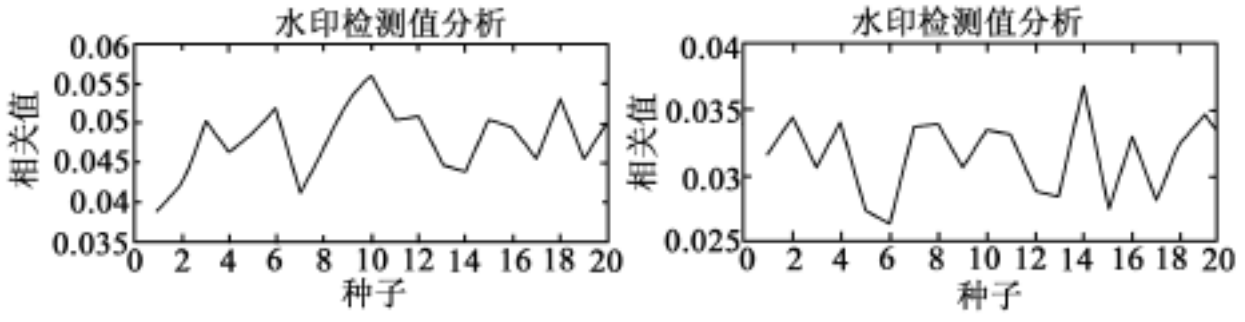

图 2.10　序列种子与水印信号

就需要了解正态随机序列是怎么得到的。关于水印与随机序列的问题将在水印的有关章节再具体叙述。这里，我们仅仅关心随机序列在信息隐藏中的一个应用方面——选择信息嵌入位。

2.6.1　随机序列与信息嵌入位的选择

一个品质良好的随机序列可以在信息安全的诸多领域发挥不可估量的作用。在信息隐藏中，最直接的一个例子就是通过随机序列控制秘密信息的嵌入规则。一个没有随机序列控制的隐藏算法是没有安全性可言的。图 2.11 是在顺序选取像素点的情况下运用 LSB 空域隐藏算法得到的效果，仔细观察不难发现在图像中隐藏有信息。

图 2.11 是由于秘密信息长度很短导致其只占用了载体图像的一部分像素位造成的。出现这样的效果基本上可以认为实验是失败了。解决这一问题的方法就是将秘密信息打散嵌入到图像中，使其不至于在一个局部形成明显的分界线。那么如何打散呢？

图 2.11　嵌入位选择不当

Kerckhoffs 原则是信息安全的一个基本原则，即系统的安全仅仅依赖于密钥而不是安全算法。同样还是举 LSB 算法的例子，如果将信息顺序地隐藏到图像中，那么将不存在密钥的应用空间。在算法公开的要求下，任何一个人都可以逐一将秘密信息提取，信息隐藏将毫无意义。那么如何使用密钥呢？

回答上述两个问题的答案就是:使用随机序列控制信息嵌入位。在整幅图像中随机选择嵌入位将解决前一个问题,而随机序列的种子当然的就可以看做密钥。

常见的随机数控制的方法有两个:随机间隔法和随机置换法。本节主要介绍随机间隔法。随机间隔法的思想比较简单,主要是利用随机数的大小控制前后两个嵌入位的距离。如一个长为 N 的服从 U(0,1) 的随机序列 $R = \{r_1, r_2, \cdots, r_N\}$, N 大于秘密信息长度。取第一个嵌入位为 i,伪 C 代码描述有:

```
inbeding address = i;
for ( j = 1; j < = length( message) ; j + + )
    {
    if ( r_j > 0. 5)
        imbeding address + = k;
     else
        imbeding address + = p;
     }
```

以上代码的实质就是通过判断相应的随机数与 0.5 的大小,若大于 0.5,则选择的嵌入位与前一个嵌入位间隔 k - 1 位,否则间隔 p - 1 位。

为了更好地在图像的二维矩阵中运用随机间隔法,我们编写了 randinterval. m 函数。该程序是一个严格意义上的随机间隔选择隐藏位的程序,今后我们将大量地用到它。对应前文所述的算法,我们这样定义了 k 与 p:

total = 图像载体总像素点;

quantity = 为要选择的像素点;

$$k = \left\lfloor \frac{total}{quantity} \right\rfloor + 1$$

p = K - 2;

函数代码如下:

```
% 文件名: randinterval. m
% 程序员: 郭迟
% 编写时间: 2004. 2. 23
% 函数功能: 本函数将利用随机序列进行间隔控制,选择消息隐藏位
% 输入格式举例: [ row, col] = randinterval( test, 60, 1983)
% 参数说明:
% matrix 为载体矩阵
% count 为要嵌入的信息的数量( 要选择的像素数量)
% key 为密钥
% row 为伪随机输出的像素行标
% col 为伪随机输出的像素列标
```

```
function [row, col] = randinterval(matrix, count, key)
% 计算间隔的位数
[m, n] = size(matrix);
interval1 = floor(m* n/count) + 1;
interval2 = interval1 - 2;
if interval2 == 0
    error('载体太小不能将秘密信息隐藏进去!');
end
% 生成随机序列
rand('seed', key);
a = rand(1, count);
% 初始化
row = zeros([1 count]);
col = zeros([1 count]);
% 计算 row 和 col
r = 1;
c = 1;
row(1, 1) = r;
col(1, 1) = c;
for i = 2: count
    if a(i) >= 0.5
        c = c + interval1;
    else
        c = c + interval2;
    end
    if c > n
        r = r + 1;
        if r > m
            error('载体太小不能将秘密信息隐藏进去!');
        end
        c = mod(c, n);
        if c == 0
            c = 1;
    end
end
row(1, i) = r;
```

```
col( 1, i)  = c;
end
```

我们在一个 8× 8 的范围内对像素进行 20 点选择, 输入:

```
>> test = zeros( 8) ;
>> [ row, col]  = randinterval( test, 20, 1983) ;
>> for  i = 1∶ 20
       test( row( i) , col( i) )  = i;
   end
```

随机间隔选择的结果如图 2.12 所示。

seed=1983　的随机序列

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.09842	0.13928	0.83403	0.47569	0.96968	0.4134	0.96501	0.84192	0.22852	0.68507	0.034194	0.70657	0.38195

Array Editor: test

File　Edit　View　Web　Window　Help

Numeric format: short G　Size: 8　by 8

	1	2	3	4	5	6	7	8
1	1		2				3	
2	4				5		6	
3			7				8	
4	9				10		11	
5			12		13		14	
6	15		16		17		18	
7	19		20					
8								

图 2.12　随机间隔选择像素

随机间隔控制法实现比较简单, 在今后的实验中我们将经常使用到。使用随机间隔控制法选择的像素位是不会发生冲突的。

2.6.2　安全 Hash 函数

在随机置换法中, 由于对随机序列不加任何的限制, 不恰当的置换策略将很容易导致选择的像素位重复出现多次, 从而造成一个像素嵌入了多个信息, 后嵌入的信息破坏了先嵌入的信息, 我们称这种情况为碰撞(collision) 。解决碰撞的机制有很多, 如数据结构中的开地址法等。这里我们将使用到安全 Hash 函数的伪随机置换算法, 该算法可以解决碰撞问题。

Hash 函数的作用是将任意长度的报文映射为一个长度固定的 Hash 码。Hash 码是报文每一位的函数。报文任意一位的改变都将导致 Hash 码的改变。常用的安全 Hash 函数有: MD5, SHA-1 等。这里, 我们使用 MD5。