

# 一种基于线性预测的无失真图像编码方案

宋振

(云南大学 信息学院)

2022 年 12 月 20 日

## 摘 要

针对传统无失真线性预测编码压缩效率不够高，和传统算术编码不适用于长信源编码问题，本文提出了一种基于线性预测的图像无失真编码方案，求解最佳系数进行线性预测，并通过分析误差取值范围，使得预测误差以更大的概率映射到零值附近，降低了描述信源所需的信息度量；并对映射后的误差，进行自适应算术编码，在不增加算法复杂度的同时，提高了编码效率。

实验得出，使用最佳 LPC 编码对图像的无损压缩率接近 50%，同在无失真前提下，最佳 LPC 编码，相较于传统 LPC 方案，压缩效率平均提高了 3.5%，且在平滑图像中，最佳 LPC 相较于经典 LPC 算法，压缩效率提升更为明显，在 cameraman 测试图像中，压缩率达到了 8.82% 的提升。

**关键词：**线性预测；最佳系数；LPC 编码；自适应算术编码；无失真图像压缩；

## 1 介绍

随着信息时代的到来,信息量爆炸式增长,数据压缩编码研究愈发重要。目前,数据压缩算法主要围绕信息熵展开研究力求高效实用的压缩算法 [1]。由于图像信息为非话业务的主要内容,且占用大量储存空间,所以要在图像通信中广泛应用数据压缩技术 [2] 特别是在版权保护、交通监控和司法鉴定等场景中,对数据的完整性有高度的要求,无损压缩就及其重要。

线性预测编码 (Linear Prediction Coding, LPC) 起初被应用于语音识别领域,在语音频谱分析、特征系数提取方面有重要应用意义 [3, 4] 在图像压缩中,利用 LPC 可以将对图像像素值本身编码转换为对像素值的预测误差进行编码,通过减小的信源符号的范围,实现更高的压缩效率。

算术编码 (Arithmetic Coding, AC) 作为继霍夫曼编码后的一大熵编码理论 [5], 直接对序列进行编码,在数据压缩中有重要应用意义。由于 AC 不适用长序列信源编码,自适应算术编码 (Adaptive Arithmetic Coding, AAC) 随之被提出 [6], 进一步推广了 AC 在数据压缩中的应用。

本文将基于 LPC 理论,使用 AAC 实现一种图像的无损压缩算法。

## 2 线性预测的无失真图像编码

### 2.1 框架概述

本文提出的基于线性预测的无失真图像编码方案框架示意图如下图 1 所示:

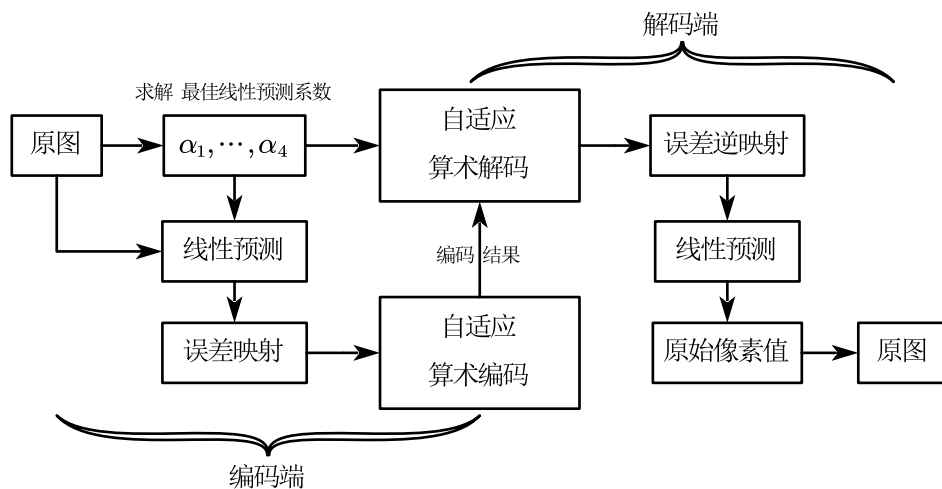


图 1: 线性预测的无失真图像编码方案框架示意图

在该编解码系统中：

编码端：对待编码压缩的图像进行遍历，并按最佳线性预测法求解出最佳预测系数；再对图像进行线性预测和误差映射，减小图像像素值的范围，即使待编码符号以较大的概率分布于零值附近，继而实现在熵编码中的较短码子表示，提高了编码效率；其中熵编码过程，基于经典的算术编码 [1]，对其算法进行改进，采用自适应概率统计法，优化了传统算术编码不适用于长度较大的信源编码问题。

解码端：目的通过编码结果，还原出压缩前的数据信息，作为编码的逆过程，与编码端工作相似但相逆，此处便不再叙述，需要注意的是，最佳线性预测解码中，需要将编码端求解的最佳预测系数同编码结果一同传到解码端。

## 2.2 线性预测编码

### 2.2.1 线性预测编码原理

设当前像素为  $x_0$ ，上一像素为  $x_1$ ，则用上一像素作为当前像素的估计得到  $x' = x_1$ ；预测估计误差为  $e = x_0 - x'$ （像素真值减掉像素估计值得到像素估计误差）；

本文利用自然图像的空间相关性，选用邻近四个像素按如下式 (1) 所示的预测方法：

$$x' = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 \quad (1)$$

其中： $x_1, x_2, x_3, x_4$  与  $x_0$  均为一幅数字图像中的像素，且  $x_1, x_2, x_3, x_4$  从时间因果关系上来说，出现在  $x_0$  之前，它们在一幅连续的图像中的位置如下图 2 所示， $x_0$  为边界像素时，取不到的  $x_i$  值为 0，预测估计误差为  $e = x_0 - x'$ 。

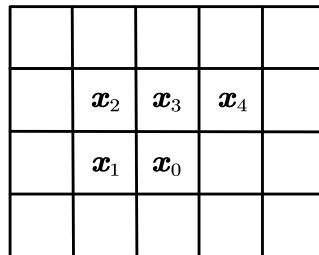


图 2: 像素位置示意图

若信号是连续变化的，则相邻像素的灰度值将较为相近。利用相邻像素对当前像素作预测后，预测误差  $e$  的概率分布将出现误差幅值在 0 附近的概率很大，而误差幅值越大，出现的机会就越小的情况，这意味着预测误差的熵较小。利用熵编码就能实现对图像的有效压缩。

### 2.2.2 利用预测值缩减预测误差取值范围

由于图像像素  $x_0$  的灰度取值范围为  $0 \sim 255$ ，预测值  $x'$  取值范围也为  $0 \sim 255$ ，预测误差的取值范围应该是  $-255 \sim 255$ ，一共 511 种不同取值，这样每个误差值的平均编码比特数将较多。

但是在考虑到预测值  $x'$  是已知的情况下，预测误差  $e = x_0 - x'$  的实际取值范围将小得多。当  $e > 0$  时，它的取值范围只能是  $0 \sim 255 - x'$ ，而当  $e < 0$  时它的取值范围只能是  $-x' \sim 0$ 。所以我们可以据此来将预测误差映射到取值范围较小的空间中。设预测误差映射值为  $E = E(e)$ 。

其映射关系如式 (2, 3) 下所示：

如果  $0 \leq x' < 128$ ：

$$E(e) = \begin{cases} -2e, & e \leq 0 \\ 2e - 1, & 0 < e \leq x' + 1 \\ e + x', & e > x' + 1 \end{cases} \quad (2)$$

如果  $128 \leq x' < 256$ ：

$$E(e) = \begin{cases} 2e, & e \geq 0 \\ -2e - 1, & x' - 256 \leq e < 0 \\ -e + x' - 255, & e < x' - 256 \end{cases} \quad (3)$$

在解码端，逆映射关系如式 (4, 5) 下所示：

如果  $0 \leq x' < 128$ ：

$$e = \begin{cases} -\frac{E}{2}, & E \leq 2x' + 1 \text{ 且 } E \text{ 为偶数} \\ \frac{E+1}{2}, & E \leq 2x' + 1 \text{ 且 } E \text{ 为奇数} \\ E - x', & E > 2x' + 1 \end{cases} \quad (4)$$

如果  $128 \leq x' < 256$ ：

$$e = \begin{cases} \frac{E}{2}, & E \leq 511 - 2x' \text{ 且 } E \text{ 为偶数} \\ -\frac{E+1}{2}, & E \leq 511 - 2x' \text{ 且 } E \text{ 为奇数} \\ 255 - x' - E, & E > 511 - 2x' \end{cases} \quad (5)$$

### 2.2.3 最佳线性预测

若能求出周围像素每一个的最佳权重，即计算出 (1) 式中各  $\alpha_i, (i = 1, \dots, 4)$  的值，称为最佳线性预测系数，并求出最佳线性预测值，则可以再次提高压缩效率。

为求解最佳线性预测系数，最佳的目标是让预测误差的平方平均值  $E[e^2]$  最小，通过求  $E[e^2]$  对所有的  $\alpha_i$  的偏导数为 0，如下：

$$\frac{\partial E[e^2]}{\partial \alpha_i} = E \left[ \frac{\partial}{\partial \alpha_i} \left( x_0 - \sum_{k=1}^4 \alpha_k x_k \right)^2 \right] = -2E[ex_i] = 0 \quad (6)$$

得：

$$-2E[x_i e] = -2E \left[ x_i \left( x_0 - \sum_{k=1}^4 \alpha_k x_k \right) \right] = 0 \quad (7)$$

化简：

$$E \left[ x_i \sum_{k=1}^4 \alpha_k x_k \right] = E[x_i x_0] \quad (8)$$

令  $R_{i0} = E[x_i x_0]$ ,  $R_{ik} = E[x_i x_k]$ ，即可将 (8) 式转换为矩阵的形式：

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ R_{21} & R_{22} & R_{23} & R_{24} \\ R_{31} & R_{32} & R_{33} & R_{34} \\ R_{41} & R_{42} & R_{43} & R_{44} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} R_{10} \\ R_{20} \\ R_{30} \\ R_{40} \end{bmatrix} \quad (9)$$

求解方程 (9) 就可以求出最佳预测系数  $\alpha_i$ 。再根据最佳预测系数再次对图像进行编码，压缩效率会得到进一步提高。

### 2.3 自适应算术编码

算术编码 AC 于 1979 年被提出 [5]，根据概率分布将信源符号映射到  $[0, 1]$  之间，其编码示意图如下图3所示：

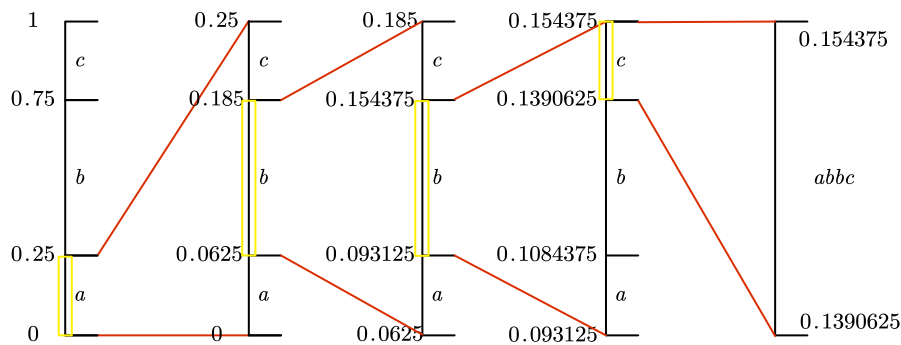


图 3: 算术编码原理说明

如图3为信源 “abbc” 的算术编码过程，根据各符号的概率分布，以及符号顺序，计算信源的分配区间，最终使用该区间内的一个数对信源进行表示（图示 “abbc” 可以由  $[0.1390625, 0.154375]$  中的任一小数表示）。

由于 AC 在编码前，需要遍历信源求解其概率分布，不适用于长信源编码问题中，于是针对任意信源长度的 AAC 算法被提出 [6]，对于信源 “*abbc*” 其编码示意图如下图4所示：

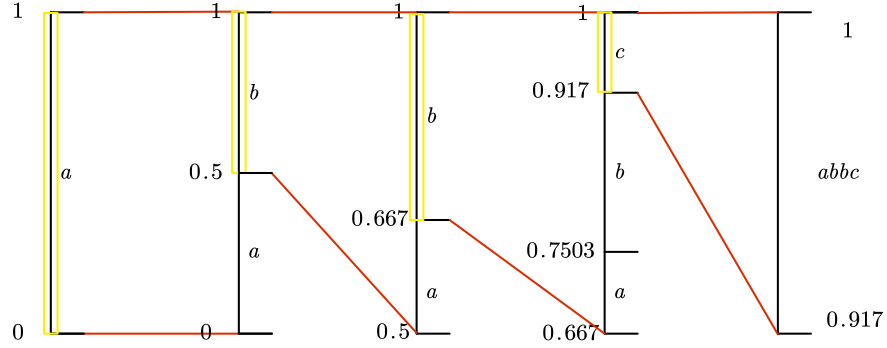


图 4: 算术编码原理说明

如图4所示，AAC 在编码前无需一直信源符号的概率分布，且无关信源长度，其区间分布取决于以编码信源的长度，相对 AC 降低了时间复杂度，且推广了适用范围。

### 3 实验

本文测试实验图像（均为  $512 \times 512$ , *.raw* 格式）如下图5所示：

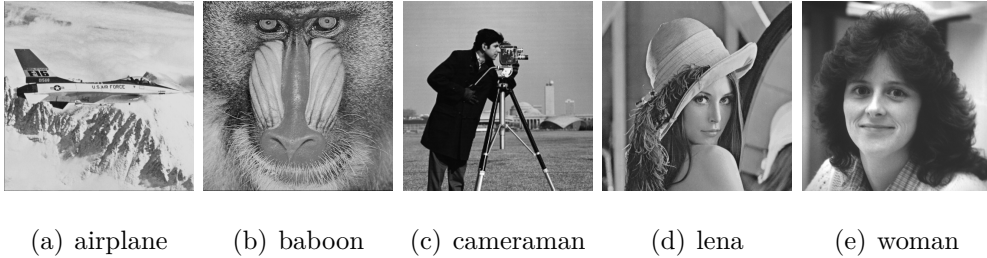


图 5: 实验测试图像集

首先对待编码图像求解最佳线性预测系数  $\alpha_1 \sim \alpha_4$ （保留四位小数），如下表1所示：

表 1: 最佳系数求解结果

	airplane	baboon	cameraman	lena	woman
$\alpha_1$	0.8882	0.6975	0.9224	0.7263	0.8896
$\alpha_2$	-0.7260	-0.1245	-0.8550	-0.5222	-0.8059
$\alpha_3$	0.8083	0.3047	0.9114	0.6616	0.8924
$\alpha_4$	0.0296	0.119	0.0209	0.1345	0.0243

其中  $\alpha_1$  和  $\alpha_3$  比较大,  $\alpha_2$  为负值, 说明在连续图像中, 像素值与其左侧和上侧像素值最接近, 与左上和右上相似较远。

各图像压缩率, 如下表2所示:

表 2: 图像压缩率结果

图像	压缩方式		
	经典 LPC	最佳 LPC	提升
airplane	43.08%	46.42%	3.34%
baboon	19.74%	22.55%	2.81%
cameraman	48.26%	57.08%	8.82%
lena	41.93%	43.16%	1.23%
woman	51.96%	52.73%	0.76%

由上表2可知, 使用最佳 LPC 对图像的无损压缩率能够接近 50%, 对于 baboon 此类高频信息更多的图像压缩率会降低, 在 cameraman 此类更为平滑的图像中压缩率最高, 说明 LPC 无损压缩更适用于平滑, 含整块相似的图像中。通过对比经典 LPC 算法和最佳 LPC 算法, 发现压缩率平均提升了 3.5%, 且在 cameraman 此类平滑图像中, 最佳 LPC 相较于经典 LPC 算法, 压缩效率提升更为明显, 达到了 8.82% 的提升。

## 4 结论

本文提出了一种基于线性预测的图像无失真编码方案, 求解最佳系数进行线性预测, 并通过分析误差取值范围, 使得预测误差以更大的概率映射到零值附近, 降低了描述信源所需的信息度量; 并对映射后的误差, 进行自适应算术编码, 在不增加算法复杂度的同时, 提高了编码效率。

实验得出, 使用最佳 LPC 编码对图像的无损压缩率接近 50%, 同在无失真前提下, 最佳 LPC 编码, 相较于传统 LPC 方案, 压缩效率平均提高了 3.5%, 且在平滑图像中, 最佳 LPC 相较于经典 LPC 算法, 压缩效率提升更为明显, 在 cameraman 测试图像中, 压缩率达到了 8.82% 的提升。

## 参考文献

- [1] K. L. Ketshabetswe, A. M. Zungeru, B. Mtengi, C. K. Lebekwe, and S. R. S. Prabhakaran, “Data compression algorithms for wireless sensor networks: A review and comparison,” *IEEE Access*, vol. 9, pp. 136 872–136 891, 2021.
- [2] 汤敏, “数据压缩技术在通信中的应用,” *无线互联科技*, vol. 19, no. 16, pp. 115–117, 2022.
- [3] R. J. Javier and Y. Kim, “Application of linear predictive coding for human activity classification based on micro-doppler signatures,” *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 10, pp. 1831–1834, Oct. 2014.
- [4] 刘鹏, 李松斌, 戴琼兴, and 邓浩江, “高效的线性预测语音编码信息隐藏方法,” *计算机工程与设计*, vol. 35, no. 04, pp. 1172–1177, 2014.
- [5] J. Rissanen and G. G. Langdon, “Arithmetic coding,” *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, Mar. 1979.
- [6] A. Moffat, “Linear time adaptive arithmetic coding,” *IEEE Transactions on Information Theory*, vol. 36, no. 2, pp. 401–406, Mar. 1990.



## A 附录概览

本文实验的程序及论文排版 L<sup>A</sup>T<sub>E</sub>X 代码见 [github: A Lossless Image Coding Based on LPC](#)，其中的自适应算术编码基于工程：Arithmetic Coding。

其他附录如下：

Matlab **最佳系数求解**：为最佳线性预测系数求解程序，使用 MATLAB 实现；

Java Main **程序**：编解码调用主程序，使用 Java 实现；

Java LPC-AAC **编码**：基于线性预测的自适应算术编码程序，使用 Java 实现；

Java LPC-AAC **解码**：基于线性预测的自适应算术解码程序，使用 Java 实现。

## B 实验程序

Listing 1: Matlab 最佳系数求解程序

```

1 %% 1、清除缓存
2 % clear;clc;
3 %% 2、读入.raw文件数据，并将数据转换为512*512的像素信息矩阵
4 row=512;column=512; %行和列
5 num=column*row; %数据总数
6
7 filename='imgs\\airplane.raw';
8 % filename='imgs\\baboon.raw';
9 % filename='imgs\\cameraman.raw';
10 % filename='imgs\\lena.raw';
11 % filename='imgs\\woman.raw';
12 fileID=fopen(filename,'r'); %只读的方式打开指定的文件
13 data=fread(fileID,num); %读入所有数据，默认为double数据类型
14 fclose(fileID); %关闭文件传输流
15 data=reshape(data,[row,column]); %重新定义数据矩阵的大小为row*
    column
16 %% 3、统计矩阵R的值
17 x=zeros(1,4); %周围像素点x1~x4矩阵
18 R_left_sum=zeros(4); %最佳线性系数方程的右侧矩阵
19 R_right_sum=zeros(4,1); %左侧矩阵
20 for i=1:row
21     for j=1:column
22         x0=data(i,j);
23         if(j==1)
24             x(1)=0;
25         else
26             x(1)=data(i,j-1);
27         end
28         if(j==1||i==1)
29             x(2)=0;

```

```

30         else
31             x(2)=data(i-1,j-1);
32         end
33         if(i==1)
34             x(3)=0;
35         else
36             x(3)=data(i-1,j);
37         end
38         if(j==row || i==1)
39             x(4)=0;
40         else
41             x(4)=data(i-1,j+1);
42         end
43         R_left_sum=R_left_sum+x.*x';
44         R_right_sum=R_right_sum+x'*x0;
45     end
46 end
47 %% 4、计算最佳线性预测权值
48 R_left=R_left_sum/num;
49 R_right=R_right_sum/num;
50 a=R_left\R_right      %求左逆矩阵与右矩阵的左乘，A\B=inv(A)*B，但比速
    度与精度都优于后者
51 %% 5、保存最佳线性预测权值到.txt文件
52 weightFileName=['weight\\',filename(7:length(filename)),'_weight.txt',
    ];
53 fid=fopen(weightFileName,'w');
54 fprintf(fid,'%4f\n',a);      %每一个数据(4位小数)单独为一行，以便于
    提取时区分数据
55 fclose(fid);

```

Listing 2: Java Main 程序

```

1  import java.io.*;
2  import java.nio.file.Files;
3
4  //主程序：LPC-ACC编解码示例及调用
5  public class main_LPC {
6      public static void main(String[] args) throws IOException {
7          for (int j = 1; j < 4; j++) {
8              int q = j;      //1,2,3,分别对应三种LPC压缩方式（1-经典
    LPC; 2-映射后LPC; 3-最佳LPC）
9              int num_symbols = (q == 1) ? 511 : 256; //选择误差量的总
    数
10             String[] inputFileArray = {"airplane.raw", "baboon.raw",
    "cameraman.raw", "lena.raw", "woman_darkhair.raw"};
11             for (String s : inputFileArray) { //对inputFileArray所
    有文件进行编解码
12                 String inputFile = "data\\imgs\\" + s;
13                 File outputFile = new File("result\\LPC_compress_" +
    q + "_" + s); //编码压缩输出
14                 File de_outFile = new File("result\\

```

```

15         LPC_decompress_of_" + q + "_" + s); //解码输出
File weightFile = new File("data\\weight\\" + s + "
    _weight.txt"); //最佳权值文件路径
16
17 //      编码
18 LPC_compress lpc_compress = new LPC_compress(
    num_symbols); //实例化最佳线性预测编码类
19 try (BitOutputStream out = new BitOutputStream(new
    BufferedOutputStream(Files.newOutputStream(
    outputFile.toPath())))) {
20     double[] a = lpc_compress.get_weight(q,
        weightFile); //获得所需的编码权值
21     lpc_compress.compress(inputFile, a, out, q);
        // 编码
22 }
23
24 //      解码
25 LPC_decompress lpc_decompress = new LPC_decompress(
    num_symbols); //解码类同编码
26 try (BitInputStream in = new BitInputStream(new
    BufferedInputStream(Files.newInputStream(
    outputFile.toPath()))); OutputStream out = new
    BufferedOutputStream(Files.newOutputStream(
    de_outFile.toPath())) {
27     double[] a = lpc_decompress.get_weight(q,
        weightFile);
28     lpc_decompress.decompress(in, a, out, q);
29 }
30 }
31 }
32
33 }
34 }

```

Listing 3: Java LPC-AAC 编码程序

```

1 import java.io.*;
2 import java.nio.file.Files;
3 import java.nio.file.Paths;
4
5 //LPC-AAC编码类
6 public class LPC_compress { //LPC:linear predictive coding 线性预
    测编码类
7     int num_symbols; //编码符号数, 多一个终止符号
8
9     LPC_compress(int num_symbol) {
10         this.num_symbols = num_symbol;
11     }
12
13     //      编码
14     public void compress(String inputFile, double[] a,

```

```

15      BitOutputStream out, int q) throws IOException { //编码方法
16          FlatFrequencyTable initFreqs = new FlatFrequencyTable(this.
17              num_symbols);
18          FrequencyTable freqs = new SimpleFrequencyTable(initFreqs);
19          ArithmeticEncoder enc = new ArithmeticEncoder(32, out);
20          int num_index = 512; //待编码数据矩阵边长
21          int[] x = new int[5]; //像素值x0~x4
22          int e, E; //误差量, e为初始误差, 用于恢复真实像素; E为映射后
23              的误差, 也是频率表统计对象
24          byte[] pixel_array = Files.readAllBytes(Paths.get(inputFile))
25              ; //一次性读入全部图像数据
26          for (int i = 0; i < pixel_array.length; i++) { //遍历编码
27              x[1] = (i % num_index == 0) ? 0 : (pixel_array[i - 1] & 0
28                  xff);
29              x[2] = (i % num_index == 0 || i < num_index) ? 0 : (
30                  pixel_array[i - num_index - 1] & 0xff);
31              x[3] = (i < num_index) ? 0 : (pixel_array[i - num_index]
32                  & 0xff);
33              x[4] = (i % num_index == num_index - 1 || i < num_index)
34                  ? 0 : (pixel_array[i - num_index + 1] & 0xff);
35              x[0] = (int) (x[1] * a[0] + x[2] * a[1] + x[3] * a[2] + x
36                  [4] * a[3]);
37              e = (pixel_array[i] & 0xff) - x[0]; //计算初始预测误
38                  差
39              E = (q == 1) ? e + 255 : mapping(e, x[0]); //根据题号选择
40                  映射规则, 第2,3问中: 将预测值取整后, 再参与映射, 以保
41                  证映射和逆映射一致, 实现无失真传输
42              enc.write(freqs, E);
43              freqs.increment(E);
44          }
45      }
46
47      // 权重读取
48      public double[] get_weight(int sel, File weightFile) throws
49          IOException { //获得预测权值
50          double[] a = new double[4]; //权值数组
51          if (sel == 3) { //第三问需要从外部文件读入权值数据
52              try (FileInputStream weightStream = new FileInputStream(
53                  weightFile);
54                  BufferedReader bufferedReader = new BufferedReader(
55                      new InputStreamReader(weightStream))) {
56                  for (int i = 0; i < 4; i++) { //按行获得权值信息,
57                      并幅值给a数组
58                      a[i] = Double.parseDouble(bufferedReader.readLine
59                          ());
60                  }
61              }
62          } else { //第一二问权值均为0.25
63              for (int i = 0; i < 4; i++) {
64                  a[i] = 0.25;
65              }
66          }
67      }

```

```

49     }
50     return a;
51 }
52
53 // 误差映射
54 public int mapping(int e, int x) { //映射函数
55     int E = 0;
56     if (x >= 0 && x < 128) {
57         if (e <= 0)
58             E = -2 * e;
59         if (e > 0 && e <= x + 1)
60             E = 2 * e - 1;
61         if (e > x + 1)
62             E = e + x;
63     } else {
64         if (e >= 0)
65             E = 2 * e;
66         if (e < 0 && e >= x - 256)
67             E = -2 * e - 1;
68         if (e < x - 256)
69             E = -(e + x - 255);
70     }
71     return E;
72 }
73 }

```

Listing 4: Java LPC-AAC 解码程序

```

1  import java.io.*;
2
3  //LPC-AAC解码类
4  public class LPC_decompress { //LPC:linear predictive coding 线性
    预测编码类
5      int num_symbols; //编码符号数, 多一个终止符号
6
7      LPC_decompress(int num_symbol) {
8          this.num_symbols = num_symbol;
9      }
10
11     // 解码
12     public void decompress(BitInputStream in, double[] a,
        OutputStream out, int q) throws IOException {
13         FlatFrequencyTable initFreqs = new FlatFrequencyTable(this.
            num_symbols);
14         FrequencyTable freqs = new SimpleFrequencyTable(initFreqs);
15         ArithmeticDecoder dec = new ArithmeticDecoder(32, in);
16         int num_index = 512; //待编码数据矩阵边长
17         int [][] pixel_array = new int[num_index][num_index]; //
            512*512
18         double[] x = new double[5]; //像素值x0~x4
19         int e, E; //误差量, e为初始误差, 用于恢复真实像素; E为映射后

```

的误差，也是频率表统计对象

```

20 for (int row = 0; row < num_index; row++) {
21     for (int column = 0; column < num_index; column++) {
22         E = dec.read(freqs);
23         x[1] = (column == 0) ? 0 : pixel_array[row][column -
24             1];
25         x[2] = (column == 0 || row == 0) ? 0 : pixel_array[
26             row - 1][column - 1];
27         x[3] = (row == 0) ? 0 : pixel_array[row - 1][column];
28         x[4] = (column == num_index - 1 || row == 0) ? 0 :
29             pixel_array[row - 1][column + 1];
30         x[0] = x[1] * a[0] + x[2] * a[1] + x[3] * a[2] + x[4]
31             * a[3];
32         e = (q == 1) ? E - 255 : inverse_mapping(E, (int) x
33             [0]); //根据题号选择映射规则，第2,3问中：将预测值
34             取整后，再参与逆映射，以保证映射和逆映射一致，实现
35             无失真传输
36         pixel_array[row][column] = e + (int) x[0];
37         out.write(pixel_array[row][column]);
38         freqs.increment(E); //在频率表统计的仍为映射后，缩小
39             范围的E；求逆映射仅为求解出原像素值
40     }
41 }
42
43 // 权重读取
44 public double[] get_weight(int sel, File weightFile) throws
45     IOException { //获得预测权值
46     double[] a = new double[4];
47     if (sel == 3) { //第三问需要从外部文件读入权值数据
48         try (FileInputStream weightStream = new FileInputStream(
49             weightFile);
50             BufferedReader bufferedReader = new BufferedReader(
51                 new InputStreamReader(weightStream))) {
52             for (int i = 0; i < 4; i++) {
53                 a[i] = Double.parseDouble(bufferedReader.readLine(
54                     ));
55             }
56         }
57     } else { //第一二问权值均为0.25
58         for (int i = 0; i < 4; i++) {
59             a[i] = 0.25;
60         }
61     }
62     return a;
63 }
64
65 // 误差逆映射
66 public int inverse_mapping(int E, int x) { //逆映射函数
67     int e = 0;
68     if (x >= 0 && x < 128) {

```

```

58         if (E <= 2 * x + 1)
59             e = (E % 2 == 0) ? E / (-2) : (E + 1) / 2;
60         if (E > 2 * x + 1)
61             e = E - x;
62     } else {
63         if (E <= 511 - 2 * x)
64             e = (E % 2 == 0) ? E / 2 : -(E + 1) / 2;
65         if (E > 511 - 2 * x)
66             e = 255 - E - x;
67     }
68     return e;
69 }
70 }
```