**长序列卷积及语音信号滤波中的定点化处理**

**实验名称：**长序列卷积及语音信号滤波中的定点化处理

**实验目的：**

1、熟悉长序列的卷积原理

2、了解语音信号的简单滤波操作

3、熟悉浮点运算的定点化计算原理

**实验原理：**

1. 长序列卷积原理：

信号通过数字系统后得到输出为：

|  |
| --- |
|  |

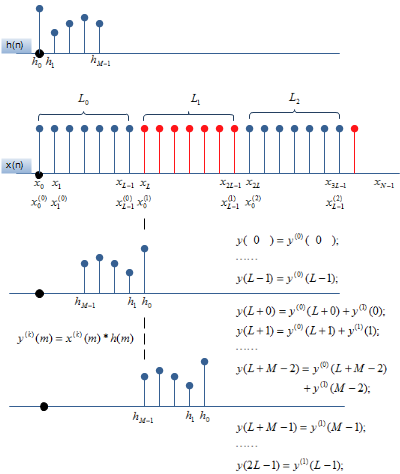
可能会很长，这样会要求大量的存储空间，而且等待输入的时间过长而导致不能实现信号的实时处理。

解决这一问题可以将分成较小的段，每一小段分别与进行卷积，然后将各小段卷积结果按一定方式“组合”在一起便可“重构”出系统相应。有两种方法：一是叠接相加法，二是叠接舍去法。

1）、叠接相加法：

如下图所示，将分为若干长度固定为*L*的小段；将与做卷积得到相应的输出，然后把按如下规律首尾相加，即可得到完整的输出：

|  |
| --- |
|  |

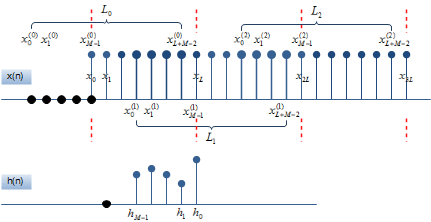


2）、叠接舍去法：

如下图所示：假设的长度为M，先在前补上M-1个0构成序列，然后将分割成一个个长度为L+M-1的小段，每相邻的两个小段有M-*1*个点是重叠的；将与做卷积得到相应的输出，然后从各中提取出中间的*L*个点拼接在一块，即可得到完整的输出：

|  |
| --- |
|  |

的首尾两段对并没有用，这些点的卷积可以完全不必计算，因此叠接舍去法相比Equation 1除了增加一些内存搬移过程不会增加任何计算量。



1. WAV文件简介：

WAV文件格式是一种音频文件格式，由微软和IBM联合开发，用于音频数字存储的标准。

3、浮点运算的整数化操作：

把浮点运算转化为整数计算可以降低计算量。做法是：对实小数进行放大求整后与整数相乘，然后将相乘结果右移适当的位数以求得正确结果，例如：

|  |
| --- |
|  |

可以按如下方式进行计算：

|  |
| --- |
|  |

在计算机中计算要比计算要快的多。

**实验内容：**

1、用Nero Wav程序打开任意一段音频，然后通过File->Save as…将其转换为”Windows PCM (\*.wav)”格式；

1. 如果是多通道的音频，通过Edit->Convert Sample Type…将多个通道合并为一个通道；

3、使用如下所示19点的FIR滤波器对音频信号进行滤波操作

|  |
| --- |
| float h[19]=  {  0.01218354, -0.009012882, -0.02881839, -0.04743239, -0.04584568,  -0.008692503, 0.06446265, 0.1544655, 0.2289794, 0.257883,  0.2289794, 0.1544655, 0.06446265, -0.008692503, -0.04584568,  -0.04743239, -0.02881839, -0.009012882, 0.01218354  }; |

4、参考代码中，滤波过程采用的是浮点的计算，需要同学们修改代码以使计算过程能够在全整数操作的情况下进行。

**编程思路：**

根据WAV的格式特点，前面的东西是文件格式自带的内容，与要处理的实际内容无关，故将frame\_start=44。

由于，需将浮点数定点化处理，故在开头处将数据乘以10^10，然后以整数运算，再再结尾处除以10^10恢复成浮点数。

**实验程序与数据：**

|  |  |
| --- | --- |
| 程序 | #include <stdio.h>  #include <string.h>  #include <math.h>  const int frameLen = 180; /\*语音帧长为点=22.5ms@8kHz3采样\*/  const int filterLen = 19; /\*滤波器长度\*/  /\*19点滤波器系数\*/  static float h[filterLen]=  {  0.01218354f, -0.009012882f, -0.02881839f, -0.04743239f, -0.04584568f,  -0.008692503f, 0.06446265f, 0.1544655f, 0.2289794f, 0.257883f,  0.2289794f, 0.1544655f, 0.06446265f, -0.008692503f, -0.04584568f,  -0.04743239f, -0.02881839f, -0.009012882f, 0.01218354f  };  short xt[frameLen+filterLen-1];  short\* const indata = &xt[filterLen-1];  short\* const pend = &xt[frameLen];  #define fseek\_success 0  #define fseek\_error 1  /\*主程序\*/  int main()  {  int i0,k0=10;  static long long int h0[filterLen];  char filename[] = "test.wav";  FILE \*fp = 0;  fp = fopen(filename,"r+b");  int frame=0;  for(i0=0;i0<filterLen;i0++)  {  h0[i0]=h[i0]\*pow(10.0,k0);  }  short outdata[frameLen];  int frame\_start=44,frame\_end;  //pcm格式的音频数据从-byte开始  fseek(fp,44,SEEK\_SET);  memset( xt, 0, sizeof(xt[0])\*filterLen );  while( frameLen==fread(indata,2,frameLen,fp) )  {  frame\_end = ftell(fp);  /\*调用低通滤波子程序\*/  for(int i=0; i<frameLen; i++)  {  long long int sum0=0;  float sum = 0;  for(int j=0; j<filterLen; j++)  sum0 += h0[j]\*indata[i-j];  sum=sum0/pow(10.0,k0);  outdata[i]=(short)sum;  }  /\*为了循环利用数组，将数据从数组尾部移到头部\*/  for(int i=0; i<(filterLen-1); i++)  xt[i] = pend[i];  /\*将滤波后的样值写入文件\*/  if( fseek\_success == fseek(fp, frame\_start, SEEK\_SET) &&  frameLen == fwrite(outdata,2,frameLen,fp) )  {  fseek(fp, frame\_end, SEEK\_SET);  frame\_start = frame\_end;  }  else  {  printf("存储结果失败\n");  break;  }  frame++;  if(frame>4000)  {  printf("手动停止\n");  break;  }  }  main\_end:  printf("一共处理了%d帧\n",frame);  if(fp)  fclose(fp);  printf("\nplease enter a char to end！\n");  getchar();  return(0);  } |
| 图像 | 捕获.JPG |

**实验总结：**

通过本实验更加深刻了解了用周期卷积计算长序列卷积的方法，以及浮点数定点化方法。了解了19点低通滤波器的工作原理，学会了用C语言设计程序进行信息处理的方法。

通过对比处理前后的音频，可以听出，处理后的音频音调明显偏低。从而证明C程序设计的低通滤波器是正确的，达到了预期效果。