数字信号处理实验一：基于 DFT 的线性卷积计算

范云潜 18373486

微电子学院 184111 班

日期：2020 年 12 月 24 日

目录

List of Figures

1 实验目的

通过使用差分方程、卷积、快速傅里叶变换等方式实现线性卷积，认识到不同方式实现的效率差异。

2 实验原理

2.1 定义法时域卷积计算

对于离散时间的卷积，其定义为 $y[n] = \sum_{k=-\infty}^{\infty} b[k]x[n-k]$ 。因此可以借助此种方法进行实现。另一方面，由于需要处理的序列为因果序列，冲激信号序列以及需要处理序列的时间从 0 开始，因此可以进行适当简化。

对于单个计算点 m 来说，将需要处理的序列 $x[n]$ 进行翻折，根据冲激序列长度 l 和 n 的关系，决定对序列的截断与 0 补充。完成后，对处理后的序列与冲激序列进行内积计算即可。

2.2 内置函数法卷积计算

直接调用内部函数实现卷积的计算，因此无需考虑其内部实现。

2.3 快速傅里叶变化法卷积计算

时域的卷积对应频域的乘法，将原始序列进行快速傅立叶变换后进行乘积后进行傅里叶反变换即可得到原始序列的卷积。

3 实验步骤

首先，输入需要进行计算的模式：

1. 差分法
2. 卷积法
3. 快速傅里叶变换法

之后根据对应的模式选择计算方式。

若是方式 1 或者 2 ，根据编写的差分法计算函数或者内建的 `conv` 函数创建一个计算 N 点序列的函数句柄，N 为两个序列长度的和减一，利用此句柄计算输出序列；若是方式 3 ，对冲激序列与输入序列进行同样点数的快速傅里叶变换，进行逐点乘积后反变换回时域即可。

4 实验代码

Listing 1: 单点卷积计算 `diff_wav.m`

```
function ret = diff_wav(x, bm, n)

% b_m= [-0.0024, -0.0042, 0.0095, 0.02, -0.038, -0.0696, 0.1374, 0.4472,
    0.4472, 0.1374, -0.0696, -0.038, 0.02, 0.0095, -0.0042, -0.0024];

b_len = length(bm);
x_inv = x(length(x):-1:1);
if (n <= length(x))
  new_x = x_inv(length(x_inv)-(n-1):length(x_inv));
else
  new_x = [zeros(n-length(x_inv),1);x_inv];
endif
leng_min = min(length(new_x),b_len);
ret = sum(new_x(1:leng_min) .* bm(1:leng_min));
```

Listing 2: 多点卷积计算 `diff_wave.m`

```
## Copyright (C) 2020 pannenetsf
##
## This program is free software: you can redistribute it and/or modify it
## under the terms of the GNU General Public License as published by
## the Free Software Foundation, either version 3 of the License, or
## (at your option) any later version.
##
## This program is distributed in the hope that it will be useful, but
## WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
## GNU General Public License for more details.
##
## You should have received a copy of the GNU General Public License
## along with this program.  If not, see
## <https://www.gnu.org/licenses/>.

## -*- texinfo -*-
## @deftypefn {} {@var{retval} =} diff_wave (@var{input1}, @var{input2})
##
## @seealso{}
## @end deftypefn

## Author: pannenetsf <pannenetsf@manjaro>
## Created: 2020-12-15

function ret = diff_wave (x, bm, n)

% val x   : the wave series
% val bm  : the response of the system
```

```
% val n   : the needed position of output, could be a single integer or an
    array

size_n = size(n, 2);
ret = zeros(1,size_n);

for i = 1:size_n
  ret(i) = diff_wav(x, bm, i);
end

endfunction
```

Listing 3: 整体计算模块 `lab1_ans.m`

```
[in_t, fs] = audioread ('./Test_music.wav');
b_m= [-0.0024, -0.0042, 0.0095, 0.02, -0.038, -0.0696, 0.1374, 0.4472, 0.4472,
    0.1374, -0.0696, -0.038, 0.02, 0.0095, -0.0042, -0.0024];
in_t = in_t(:);
b_m = b_m(:);
prt = 1

t=input("please input mode: 1) diff 2)conv 3)DFT\n");
N = length(in_t) + length(b_m) - 1;

if (t==1)
  func_comp = @(x, h) diff_wave(x, h, 1:N);
elseif (t==2)
  func_comp = @(x, h) conv(x, h);
endif

if (t == 2 && prt == 1)
    figure(6);
    stem(b_m);
    title("The time zone waveform of h");
    saveas(6, './l1s1p6.png');
endif

if (t == 3)
  [H, W] = freqz(b_m, 1, [0:2*pi/400:2*pi]);
  H_abs=abs(H);
  H_ang=angle(H);
  in_fft = fft(in_t, N);
  h_fft = fft(b_m, N);
  out_fft = in_fft .* h_fft;
  in_fft = fftshift(in_fft);
  h_fft = fftshift(h_fft);
  out_t = ifft(out_fft);
```

```
  out_fft = fftshift(out_fft);
  w = 0:2*pi/(N-1):2*pi;
  if (prt == 1)
    figure(3);
    subplot(2,1,1)
    stem(H_abs)
    title("The frequency response of abs(H)");
    subplot(2,1,2)
    stem(H_ang)
    title("The frequency response of angle(H)");
    figure(4);
    stem(abs(in_fft));
    title("The frequency domain waveform of INPUT");
    figure(5);
    stem(abs(out_fft));
    title('The frequency domain waveform of OUTPUT');

    saveas(3,'./l1s1p3.png');
    saveas(4,'./l1s1p4.png');
    saveas(5,'./l1s1p5.png');
  endif
else
  out_t = func_comp(in_t, b_m);
endif

if (prt == 1)
  figure(1);
  stem(in_t);
  title('The time domain waveform of INPUT');
  figure(2);
  stem(out_t);
  title('The time domain waveform of OUTPUT');
  saveas(1,'./l1s1p1.png');
  saveas(2,'./l1s1p2.png');
endif


input("Enter to Play origin wave...\n")
if (prt == 1)
  close all force;
endif

player_ori = audioplayer(in_t, fs);
play(player_ori)

input("Enter to Play processed wave...\n")
player_prc = audioplayer(out_t, fs);
```

```
play(player_prc)

input("Enter to Continue...")
```

5 实验结果

5.1 定义法时域卷积计算

运行 lab01_ans.m 选择模式 1 。

结果如 **图 1**，**图 2** 。

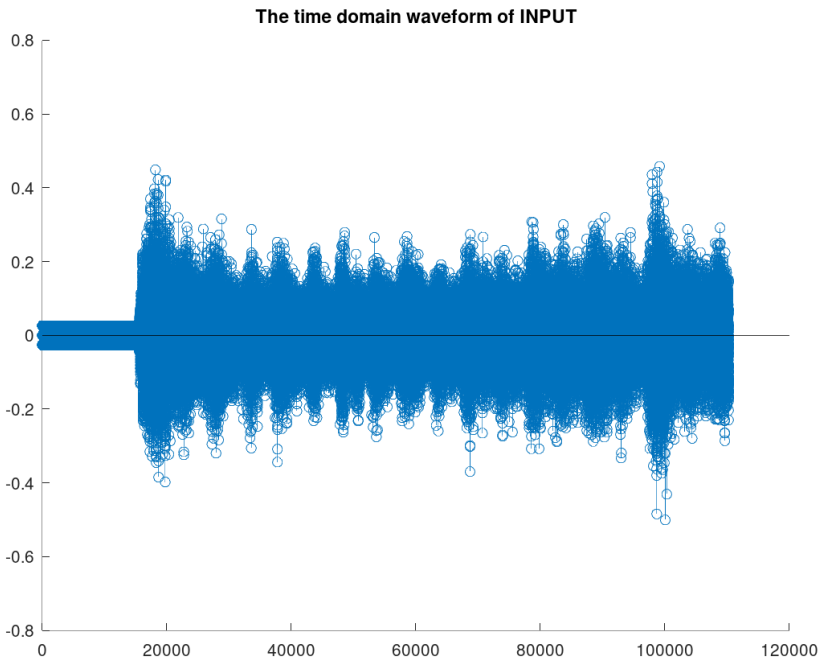

图 1: 输入音频信号的时域显示

5.2 内置函数法时域卷积计算

将上一步的 diff_wave 函数转换为 conv 函数即可。

运行 lab01_ans.m 选择模式 2 。

结果如 **图 3**，**图 4**，**图 5** 。

5.3 快速傅里叶变换实现线性卷积计算

运行 lab01_ans.m 选择模式 3 。

结果如 **图 6**，**图 7** ，**图 8** 。

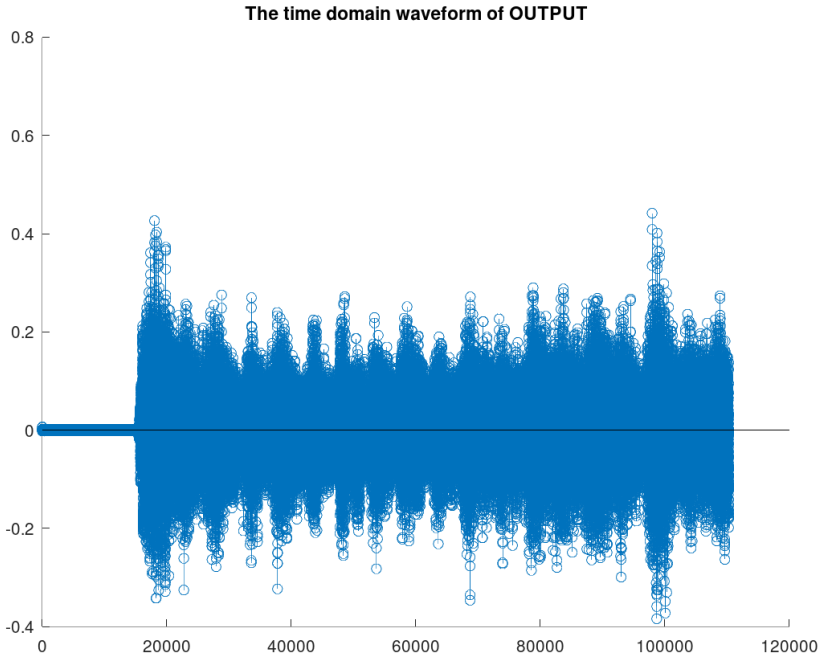

图 2: 输出音频信号的时域显示

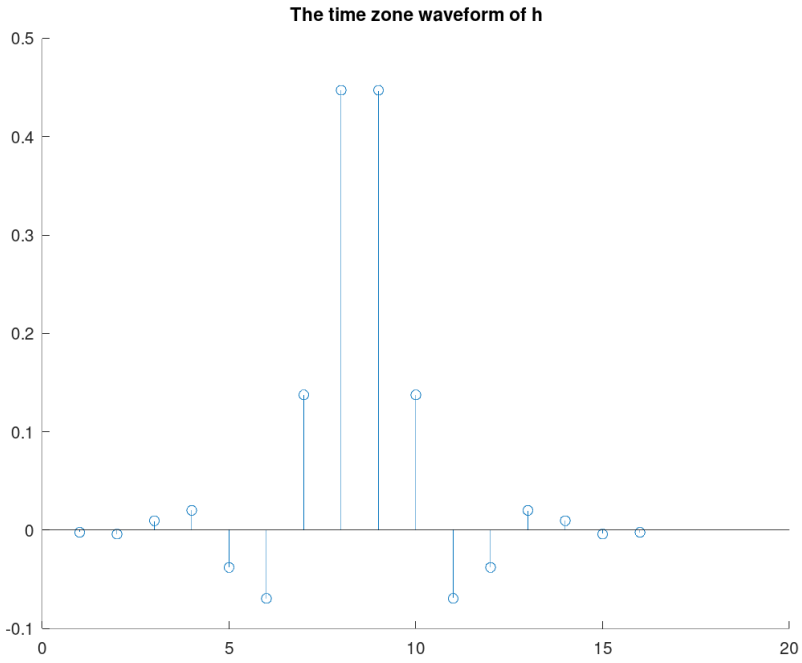

图 3: 系统函数的时域显示

6 结果分析

6.1 声音处理

用到了 `fft` 和 `ifft` 函数，播放时发现只有尖锐或者厚重的声音较为明显，符合低通滤波特性。

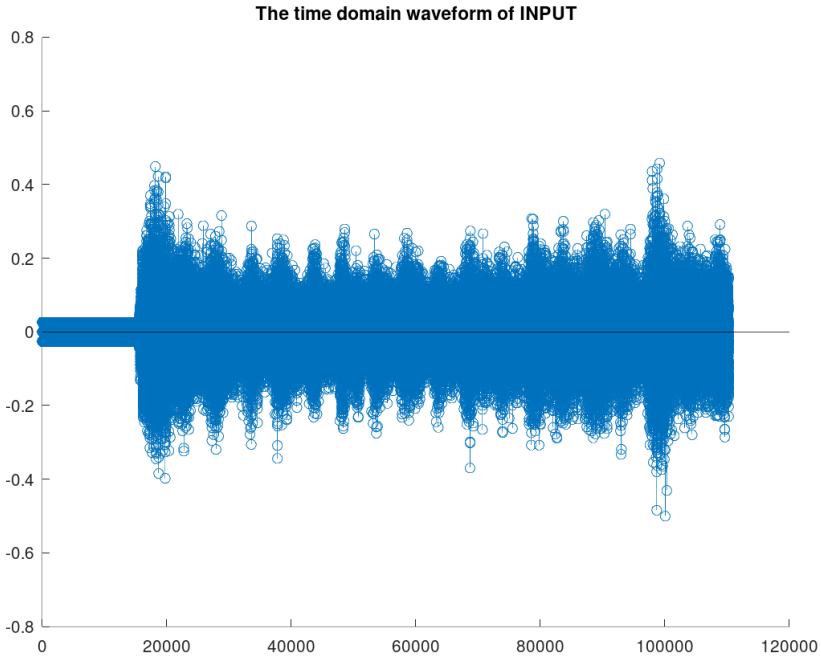

图 4: 输入音频信号的时域显示

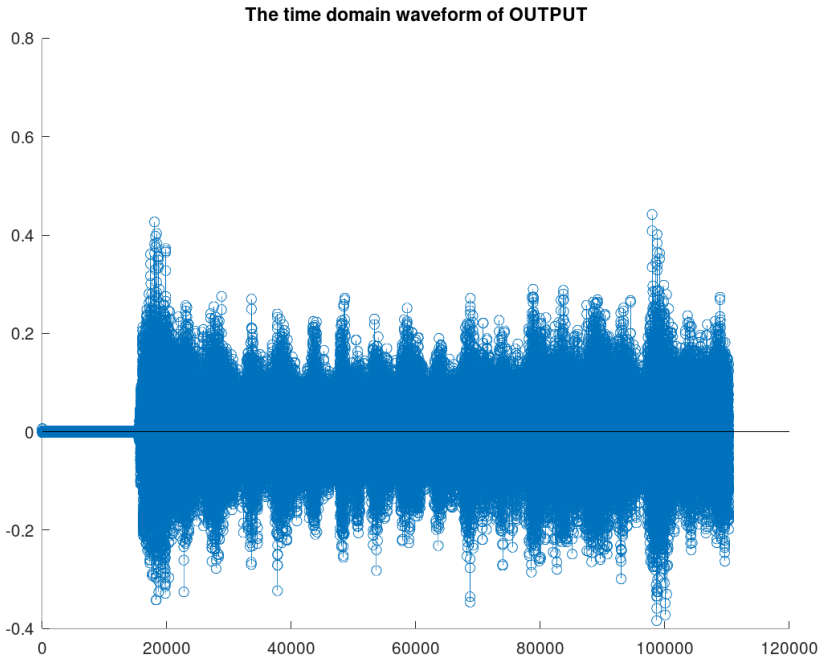

图 5: 输出音频信号的时域显示

为了更明显的表现出这几种处理方式的效果进行频谱的比较，如 **图 9** 。可以看出，这几种方式是等价的。

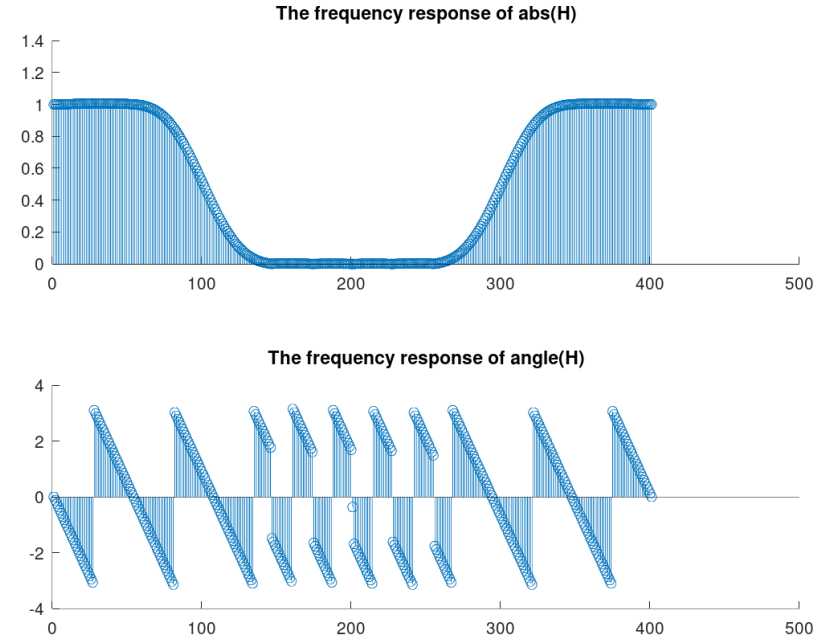

图 6: 系统函数频域显示

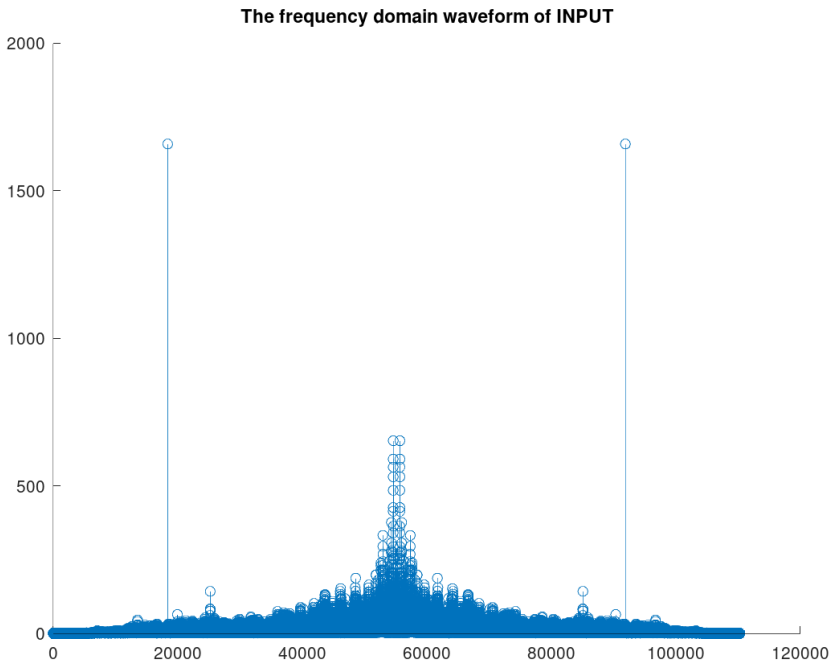

图 7: 输入波形幅频显示

6.2 思考题目：FFT 的点数选择

为了使用 FFT（圆周卷积）完成线性卷积，需要使得圆周可以覆盖整个卷积输入的区间，因此点数需要满足 $N \geq L_1 + L_2 - 1$ ， L 为时域序列的长度。

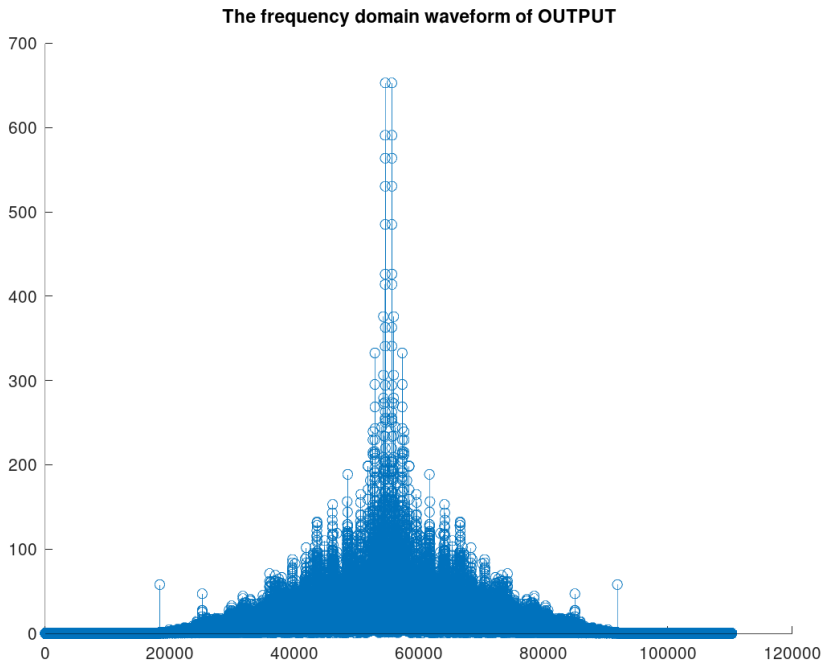

图 8: 输出波形幅频显示

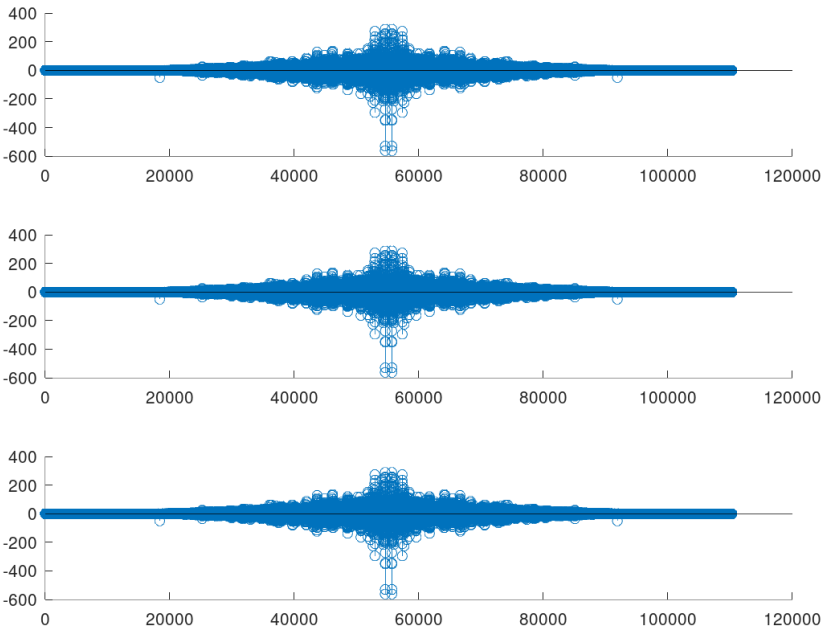

图 9: 处理方式的比对