第3章

第3章 离散时间信号的傅里叶变换

3.1 CTFS， CTFT

3.2 DTFT

3.3 CT信号的抽样

3.4 DTFS，DFS

为了引出DFT

3.5 DFT 重点内容

3.6 用DFT计算线性卷积

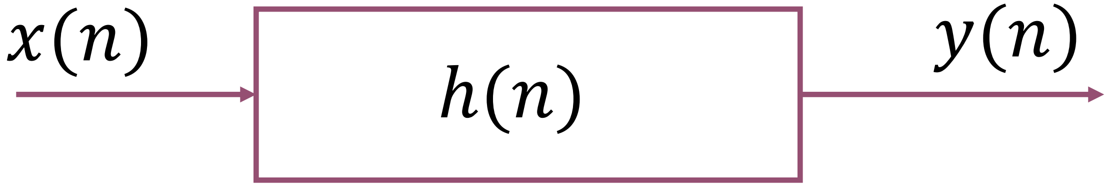
3.7 与DFT有关的几个问题

3.8 二维傅里叶变换

3.9 Hilbert 变换

傅立叶变换是信号分析与处理的基本工具

长序列卷积的计算



数字信号处理的优势是“实时实现”：信号进来后，经处理，要立即输出出去。

根据：

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

若$x(n)$没有全部进入，如何实现卷积?

若全部进入再卷积，又如何保证实时实现?

按点处理；按帧处理；案例分析

例如：FIR系统，$h(n)$有限长（例如长度在20 ~ 50左右，$x(n)$**可能很长**，也**不适宜直接卷积**。对于IIR系统， $h(n)$有限长，但差分方程描述的系数是有限个。

关键是将$x(n)$**分段**和$h(n)$**卷积**

$$
\begin{aligned}
x(n):& \quad N \\
h(n):& \quad M \\
y(n):& \quad N+M-1
\end{aligned}
$$

将$x(n)$分成L段，**每段长度**：$K=N/L$

$$
\left.\begin{array}{c}
x_1(n), x_2(n), \cdots, x_L(n) \\
y_1(n), y_2(n), \cdots, y_L(n) \\
K+M-1
\end{array}\right\}
\quad
\begin{aligned}
&L(K+M-1) \\
&=N+LM-L \\
&\neq N+M-1
\end{aligned}
$$

可以采用的方法:

Overlap — add method 叠接相加法

Overlap — save method 叠接舍去法

掌握程序设计

叠接相加法分析（1）

分段计算：

$$h(n), 0 \le n \le M-1, M\text{点}$$

$$x_1(n), 0 \le n \le L-1, L\text{点}; y_1(n), 0 \le n \le L+M-2, L+M-1\text{点}$$

$$x_2(n), L \le n \le 2L-1, L\text{点}; y_2(n), L \le n \le 2L+M-2, L+M-1\text{点}$$

从 $y_1(n)$ 、$y_2(n)$ 可以看出，其自变量有重叠部分：$L \le n \le L+M-2$，长为 $M-1$ 点

分析重叠点上各段输出间的关系：

计算 $y_1(n)$ 时，假定输入为 $x_1(n)$，$0 \le n \le L-1$，其余点上输入为 0。事实上不是。并导致在 $L \le n \le L+M-2$ 点上，$y_1(n)$ 计算不完整，因为它忽略了 $n \ge L$ 后的输入。

计算 $y_2(n)$ 时，假定输入为 $x_2(n)$，$L \le n \le 2L-1$，其余点上输入为 0。事实上不是。并导致在 $L \le n \le L+M-2$ 点上，$y_2(n)$ 计算不完整，因为它忽略了 $n < L$ 上的输入。

事实上，在 $L \le n \le L+M-2$ 点上，$y(n) = y_1(n) + y_2(n)$，这就是叠接相加法。

结论：上一段的后过渡过程与本段的前过渡过程的对应点相加。$M-1$ 点对应相加。

叠接相加法分析（2）

n	$y(n)=x(n)*h(n)$	$y_1(n)=x_1(n)*h(n)$	$y_2(n)=x_2(n)*h(n)$
0	$y(0)=h_0x_0$	$y_1(0)=h_0x_0$	
1	$y(1)=h_1x_0+h_0x_1$	$y_1(1)=h_1x_0+h_0x_1$	
2	$y(2)=h_2x_0+h_1x_1+h_0x_2$	$y_1(2)=h_2x_0+h_1x_1+h_0x_2$	
3	$y(3)=h_2x_1+h_1x_2+h_0x_3$	$y_1(3)=h_2x_1+h_1x_2+h_0x_3$	
4	$y(4)=h_2x_2+h_1x_3+h_0x_4$	$y_1(4)=h_2x_2+h_1x_3+h_0x_4$	
5	$y(5)=h_2x_3+h_1x_4+h_0x_5$	$y_1(5)=h_2x_3+h_1x_4$	$y_2(5)=h_0x_5$
6	$y(6)=h_2x_4+h_1x_5+h_0x_6$	$y_1(6)=h_2x_4$	$y_2(6)=h_1x_5+h_0x_6$
7	$y(7)=h_2x_5+h_1x_6+h_0x_7$	$y_1(7)=0$	$y_2(7)=h_2x_5+h_1x_6+h_0x_7$
8	$y(8)=h_2x_6+h_1x_7+h_0x_8$		$y_2(8)=h_2x_6+h_1x_7+h_0x_8$
9	$y(9)=h_2x_7+h_1x_8+h_0x_9$		$y_2(9)=h_2x_7+h_1x_8+h_0x_9$
10	$y(10)=h_2x_8+h_1x_9+h_0x_{10}$		$y_2(10)=h_2x_8+h_1x_9$
11			$y_2(11)=h_2x_9$
12			$y_2(12)=0$

Matlab函数实现：filter、filtic
MATLAB快速计算：fftfilt
设计程序C语言程序实现：filter

分析另一方法：补充数据，合理的循环程序

长语音实时滤波，C语言浮点子程序定点化

若存储空间有限，输入数据要分两段来做，先对第一个段数据滤波，然后对第二段数据滤波，需要对前一段信号滤波的最后条件作为后一段信号滤波的初始条件。仅用filter函数可以实现。也可以用filtic函数为filter函数提供初始条件。这个方法可用于实现非零初始条件差分方程的求解。LTIDTS可用差分方程描述，然后用线性卷积实现输出，可以视为滤波操作。线性卷积可以用循环卷积来实现。

示例代码:

```
a = [1 2 3 4 5 4 3 2 1 1 1 2 3 2 1 3];
b = [2 3 0 4 3 2];
c = conv(a, b);
[d zf] = filter(b,1,a(1:8))
e = filter(b, 1, a(9:16), zf)
c =
2   7   12   21   33   43   47   51   47   39
30  23  23   22   21   29   30   16   19   11
6
d =
2   7   12   21   33   43   47   51
e =
47  39  30   23   23   22   21   29
```

循环卷积既可以在时域直接实现，也可以在频域借助于DFT计算实现。DFT有快速算法，当信号长度N很大时，频域计算速度要比时域计算速度快很多，从而实现快速线性卷积。

MATLAB函数fftfilt基于FFT实现长序列线性卷积之重叠相加法，属于频域滤波，仅对FIR有效；fftfilt(b,x)计算结果等价于filter(b,1,x)。

时域滤波与频域滤波

filter：1 维数字滤波器

时域滤波的方法有很多

语法

y = filter(b,a,x)
y = filter(b,a,x,zi)
y = filter(b,a,x,zi,dim)
[y,zf] = filter(___)

说明

y = filter(b,a,x) 使用由分子和分母系数b和a定义的有理传递函数 对输入数据 x 进行滤波。若 a(1) 不等于1，则filter按a(1)对滤波器系数进行归一化。因此，a(1) 必须是非零值。

如果 x 为向量，则 filter 将滤波后数据以大小与 x 相同的向量形式返回。

如果 x 为矩阵，则 filter 沿着第一维度操作并返回每列的滤波后的数据。

如果 x 为多维数组，则 filter 沿大小不等于 1 的第一个数组维度进行计算。

y = filter(b,a,x,zi) 将初始条件 zi 用于滤波器延迟。zi 的长度必须等于 max(length(a),length(b))-1。

y = filter(b,a,x,zi,dim) 沿维度 dim 进行计算。例如，如果 x 为矩阵，则 filter(b,a,x,zi,2) 返回每行滤波后的数据。

[y,zf] = filter(___) 还使用任一上述语法返回滤波器延迟的最终条件 zf。

利用filter函数实现分段卷积

%数据分成4段
a = [1 2 3 4 5 4 3 2 1 1 1 2 3 2 1 3 3 4 6 8];
b = [2 3 0 4 3 2];
c = conv(a, b)
c = 1×25
2 7 12 21 33 43 47 51 47 39 30 23 23 22 21 29
36 33 43 57 55 42 58 36 16

[d **zf**] = **filter**(b,1,a(1:5))
d = 1×5 % 输出
2 7 12 21 33 % 输出

zf = 5×1：35 29 38 23 10 % 输出

[e **zf**] = **filter**(b, 1, a(6:10), zf)
e = 1×5 % 输出
43 47 51 47 39 % 输出

zf = 5×1：28 16 11 5 2 % 输出

[f **zf**] = **filter**(b, 1, a(11:15), zf)
f = 1×5
30 23 23 22 21

zf = 5×1：23 21 16 7 2

[g **zf**] = **filter**(b, 1, a(16:20), zf)
g = 1×5
29 36 33 43 57

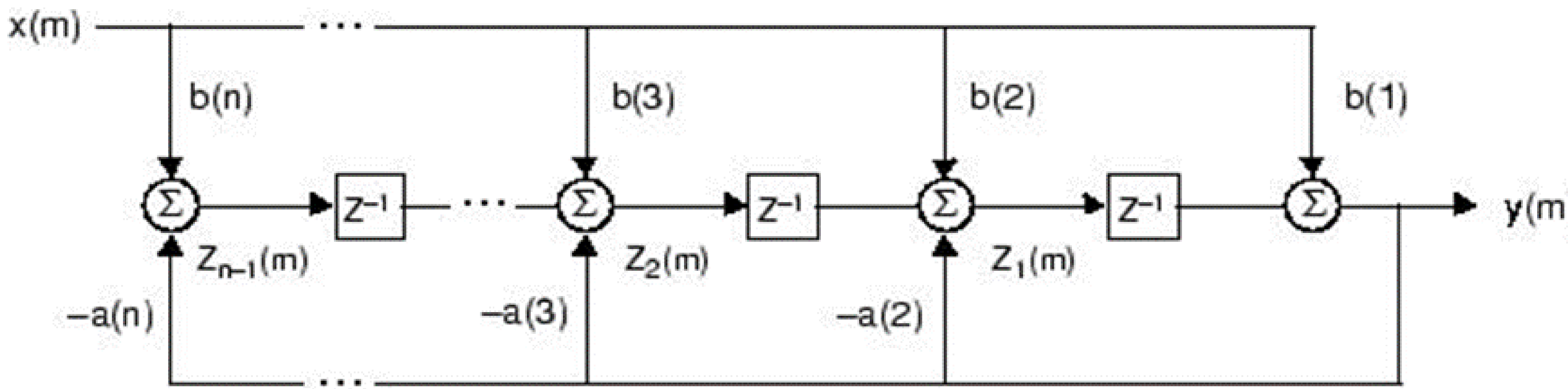
zf = 5×1：55 42 58 36 16

h = [d e f g]
h = 1×20 % 输出
2 7 12 21 33 43 47 51 47 39 30 23 23 22 21 29
36 33 43 57

c % 输出在这里，便于查看比较
c = 1×25
2 7 12 21 33 43 47 51 47 39 30 23 23 22 21 29
36 33 43 57 55 42 58 36 16

i = sum(c(1:length(h))-h) %简单验证
结果：i = 0

filtic和filter



$$
\begin{aligned}
y(m) &= b(1)x(m) + z_1(m-1) \\
z_1(m) &= b(2)x(m) + z_2(m-1) - a(2)y(m) \\
\vdots\ &= \ \vdots \qquad\qquad \vdots \\
z_{n-2}(m) &= b(n-1)x(m) + z_{n-1}(m-1) - a(n-1)y(m) \\
z_{n-1}(m) &= b(n)x(m) - a(n)y(m)
\end{aligned}
$$

初始条件z_i

从线性常系数差分方程来理解，结合直接II型结构

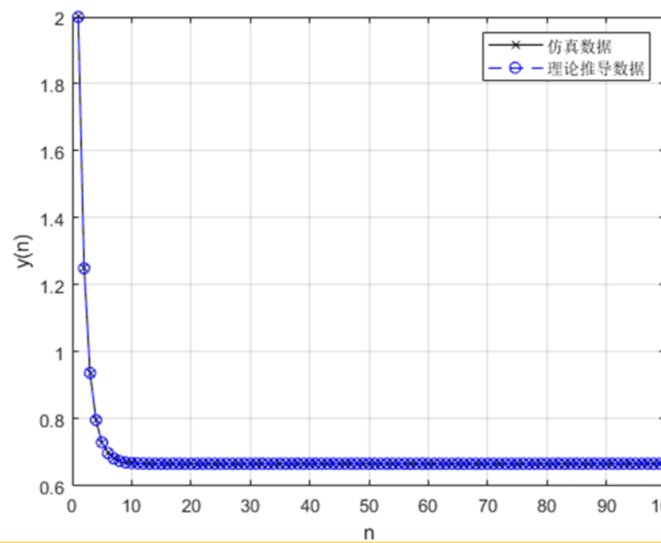
对于初始松弛系统，系统启动之前，$z_i(1)\ldots$

某 LTI 系统由下列差分方程描述 $y(n)-\frac{3}{2}y(n-1)+\frac{1}{2}y(n-2)=x(n)$，$n\geq 0$，若系统的初始条件为 $y(-1)=4$ 和 $y(-2)=10$，求系统对信号 $x(n)=4^{-n}u(n)$ 的响应。

初始状态不为零，可调用 filter(b,a,x,xic)来实现，其中 xic 为系统的等效初始状态输入数组。为了求得等效初始状态，可调用 xic=filtic(v,a,Y,X)实现，其中 Y、X 为系统的初始状态。

$$a^n u(n) \xrightarrow{\text{ZT}} \frac{1}{1-az^{-1}}$$

```
clear; close all;    %示例代码
b = [1,0,0]; a = [1,-3/2,1/2];    % 系统函数的分子、分母系数
Y = [4,10]; X=0;                  % 系统的初始条件
xic = filtic(b,a,Y,X)             % 求等效初始状态
xic = 1×2                         % 输出
     1    -2
n = 0:99; x = (1/4).^n;
y = filter(b,a,x,xic);
plot(y,'k-x')
xlabel('n'); ylabel('y(n)'); hold on
t = (1/2).^n+2/3+1/3*(1/4).^n;
plot(t,'b--o')
legend('仿真数据','理论推导数据');grid on
hold off
```



设计程序C语言程序实现：filter

时域滤波与频域滤波

频域滤波的
方法有很多

借助于**fft**的函数（DFT的快速实现）
在频域内实现一般IIR滤波的一般方法是：

```
n = length(x);
y = ifft(fft(x).*fft(b,n)./fft(a,n)); %b, a为滤波器系数
```

计算结果与filter函数的计算结果几乎相同，注意边缘效应；
对长序列很不方便，因为fft变换要很多补零操作，更复杂。

对于FIR滤波器，又若输入信号是长序列，则可以截断成短序列

```
y = fftfilt(b,x,M)
```

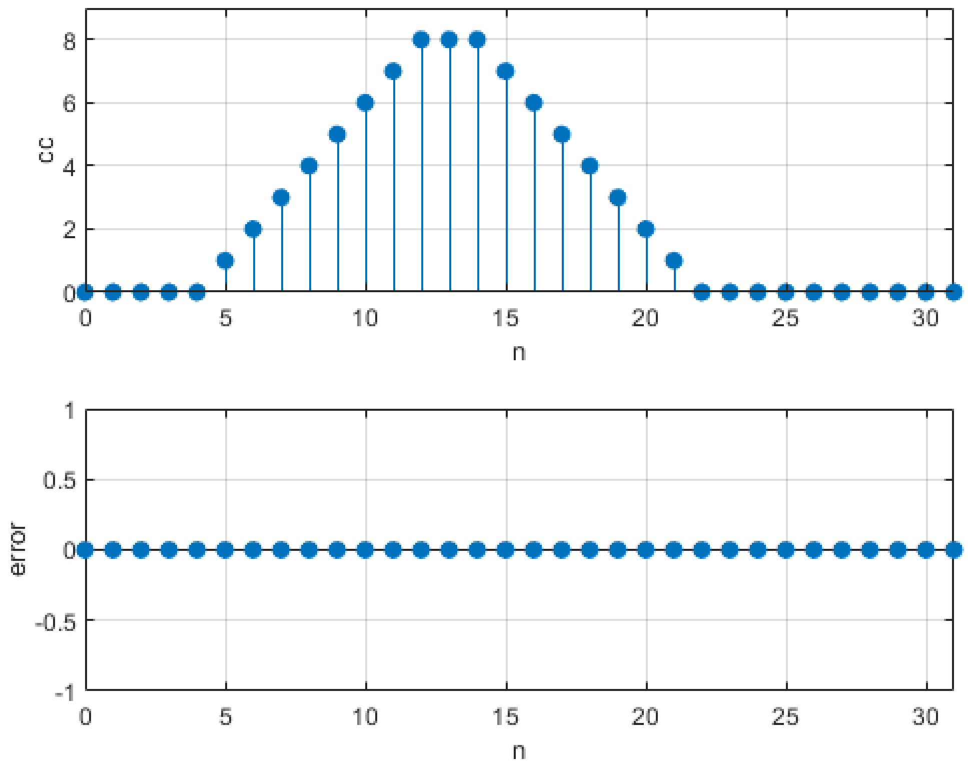
方便FFT变换操作，重叠相加法；
b，系统的单位脉冲响应；
a，输入序列向量；
M，输入序列的分段长度，默认为512。

FFT：DFT的快速算法
IFFT：IDFT快速计算
fft函数
ifft函数

直接卷积和快速卷积

利用conv指令直接计算卷积，以及利用fft和ifft指令快速计算卷积

```
clear; close all;    %示例代码
a = ones(1,13);a([1,2,3])=0;           % 时间序列从0时刻开始算起
b = ones(1,10);b([1,2])=0;             % 时间序列从0时刻开始算起
c = conv(a,b);                         % 直接卷积
M = 32;                                % fft-ifft计算卷积
AF = fft(a,M); BF = fft(b,M);
CF = AF.*BF;                           % 从0时刻开始绘图
c1 = ifft(CF);
cc = real(ifft(CF));           % 过滤掉由于截断误差引起的虚部
n = 0:(M-1);
c(M) = 0;                      % c的尾部补零，与cc同长度!
error = c-cc;                  % 两种卷积方法的差值
subplot(211); stem(n, cc, 'filled');grid;axis([0,31,0,9]);
xlabel('n'); ylabel('cc');
subplot(212); stem(n, error, 'filled');grid;axis([0,31,-1,1]);
xlabel('n'); ylabel('error');
```



c = 0 0 0 0 0 1 2 3 4 5 6 7 8 8 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0

cc =0.0000 0 0.0000 0.0000 0.0000 1.0000 2.0000 3.0000 4.0000 5.0000 6.0000
7.0000 8.0000 8.0000 8.0000 7.0000 6.0000 5.0000 4.0000 3.0000 2.0000 1.0000
0.0000 0.0000 0 0 0.0000 0 0.0000 -0.0000 0.0000 0.0000

cc==c1

ans = 1×32 logical 数组

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

filter和fftfilt 一个比较

fftfilt and filter for Short and Long Filters

Verify that filter is more efficient for smaller operands and fftfilt is more efficient for large operands.

Filter 10^6 random numbers with two random filters: a short one, with 20 taps, and a long one, with 2000.

Use tic and toc to measure the execution times. Repeat the experiment 100 times to improve the statistics.

```
clear; close all;    %示例代码
rng default
N = 100;
shrt = 20;
long = 2000;
tfs = 0;
tls = 0;
tfl = 0;
tll = 0;
for kj = 1:N
    x = rand(1,1e6);
    bshrt = rand(1,shrt);
    tic
    sfs = fftfilt(bshrt,x);
    tfs = tfs+toc/N;
    tic
    sls = filter(bshrt,1,x);
    tls = tls+toc/N;
    blong = rand(1,long);
    tic
    sfl = fftfilt(blong,x);
    tfl = tfl+toc/N;
    tic
    sll = filter(blong,1,x);
    tll = tll+toc/N;
end
```

```
tfs =
0.0842

tls =
0.0052

tfl =
0.0478

tll =
0.0620
```

长语音信号实时滤波

1. 主程序

（1）正确读取（wav）数据文件和数据

（2）按帧读取数据、调用滤波子程序；每帧数据量，数据数据类型

可以借助于MATLAB验证数据正确

为什么要分帧？帧移？

2. 子程序

（1）要求：实现长语音信号的分段卷积（滤波）

（2）要求：浮点子程序→定点化实现

（3）对定点化数据的理解

（4）分段卷积代码的正确理解，和

（5）改编（主要改变数据类型，以及相关操作的处理）。验证：

3. 滤波器（已经给定，系数为纯小数，？）

（1）分析滤波器的频率特性：低通、高通、带通？通带频率？

（2）滤波器系数定点化

可用MATLAB做频谱分析图

4. 信号频谱分析

（1）整个语音信号的频谱图

（2）语谱图

（3）找到3~5个连续语音帧并绘制各帧信号的频谱

语谱图帧移？

可以借助于Cool Edit对比频谱图和语谱图

5. 加低频和高频正弦波噪声，滤波后保留低频正弦波、滤除高频正弦波

（1）分析含躁语音信号和去噪语音信号的频谱，操作为4中的3个步骤

长语音实时滤波

C语言浮点子程序定点化

对基本C源代码理解、改编、深入分析

验证结果：可在Cool Edit和MATLAB中播放语音数据/文件

Cool Edit Pro

1. 录音

录制wav文件的步骤：录音前要设置（采样率、声道、分辨率），然后录音，最后停止录音并保存文件。保存文件时可以看到有18种文件格式供选择，可以选择8kHz采样、单声道、16-bit、Window PCM (.wav)

2. 播放

可以是原始WAVE文件或处理之后的非WAVE格式数据文件，播放WAVE文件时要合理选择参数：采样率、声道、分辨率。

3. 处理

对比滤波前后语音信号的时域波形和语谱图，尤其是可以从语谱图上看到低通滤波器的效果以判断程序做得是否合理。

4. WAVE文件

WAVE文件是计算机领域最常用的数字化声音文件格式之一，它是微软专门为Windows系统定义的波形文件格式（Waveform Audio），扩展名为"*.wav"。

WAVE文件数据本身的格式为PCM或压缩型。所有的WAV都有一个文件头，这个文件头音频流的编码参数。

WAVE文件有很多不同的压缩格式，正确而详细地了解各种WAVE文件的内部结构是成功完成压缩和解压缩的基础，也是生成特有音频压缩格式文件的前提。

最基本的WAVE文件是PCM（脉冲编码调制）格式的，这种文件直接存储采样的声音数据没有经过任何的压缩，是声卡直接支持的数据格式，要让声卡正确播放其它被压缩的声音数据，就应该先把压缩的数据解压缩成PCM格式，然后再让声卡来播放。

5. PCM数据格式

PCM(Pulse Code Modulation)也被称为 脉码编码调制。PCM中的声音数据没有被压缩，如果是单声道的文件，采样数据按时间的先后顺序依次存入。它的基本组织单位是BYTE(8bit)或WORD(16bit)