3 Matlab 综合应用案例

课前学习任务

（1）安装小波分析工具箱 Wavelet Toolbox 和图像处理工具箱 Image Processing Toolbox。

（2）阅读课件和讲义，基本理解信号时频分析原理，调试运行课件中涉及到的函数在 help 中的案例程序。

3.1 信号频谱分析之傅里叶分解

3.1.1 基础知识

信号的频谱分析是我们认识信号非常重要的方法，很多时候我们在时域里看不出信号的特征，但当我们做傅里叶分解得到频谱后，便能一目了然。频谱分析还是我们做信号处理比如说压缩、滤波等的重要依据。在微积分和电路理论课程中我们已经学习了傅里叶分解，正变换公式为

$$\hat{x}_1(\omega) = \hat{x}\left(\frac{\omega}{2\pi}\right) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}\,dt$$

逆变换公式为

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{x}_1(\omega)e^{it\omega}\,d\omega$$

我们对傅里叶分解比较熟悉的是，信号可以分解成多个正弦信号的叠加，每个正弦信号的幅值、频率和相位可以由傅里叶变换公式计算得到。这里从另外一个角度介绍一下傅里叶分解，为后面小波分析做点铺垫。

傅里叶分解实际上是不断地用信号与不同频率和相位的正弦函数做乘积并积分（可认为是内积运算，也可看做是带通滤波），得到各个频率和相位下的系数，即傅里叶变换结果，利用这组结果可以实现信号的重建。这个过程实际上可认为是基于函数空间理论的，即：**正弦函数是有界连续函数空间中一组完备的基函数，也就是说，任何一个函数由一系列正弦函数叠加而得，每个正弦函数前的系数就是所谓的空间坐标**。实际上支撑（张成）连续函数空间的基函数有很多组，对不同的基函数取法，信号函数的坐标特点和复杂程度也各不相同。就像我们所处的三维欧几里得空间一样，我们可以取一组互相垂直且为单位长度的矢量($\mathbf{e_x}$, $\mathbf{e_y}$, $\mathbf{e_z}$)为基，这时任意一个向量比如 OA(3,2,5)，$OA=3\mathbf{e_x}+2\mathbf{e_y}+5\mathbf{e_z}$，$x=3=OA\cdot \mathbf{e_x}$（点乘就是一种内积（投影）运算）,(3,2,5)称为 OA 再这个坐标系下的坐标。也可以选择不互相垂直的矢量组作为基（只要互相不共线即可）。所以说，**傅里叶**

分解过程相当于是在不断求取带分解函数在由正弦函数张成的函数空间中的坐标的过程。

傅里叶分解的解析计算再 matlab 中课通过调用 fourier()函数实现，案例参考 help 文档。能进行解析傅里叶分解的函数有限，实际上，我们之所以使用 matlab 进行计算，大多是难以解析计算的情况。同时，符号计算并不是 Matlab 的强项，符号计算功能更强的软件有 Mathematica 等。

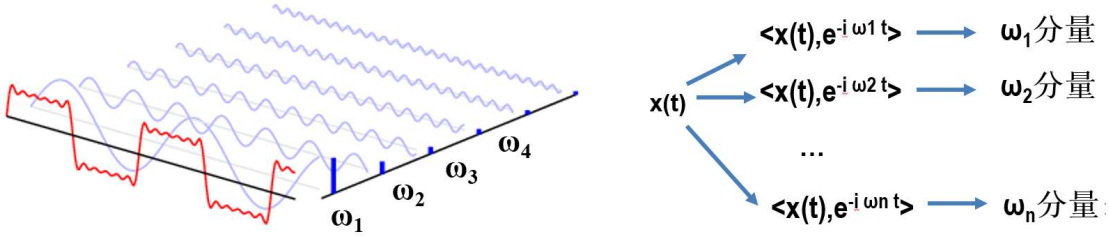

图 3.1 傅里叶分解原理

3.1.2 快速傅里叶分解

对我们获取到的一段信号，总是有限长的一段数据，然而我们傅里叶变换的公式中的积分区间却是-∞到+∞，怎么办呢？我们能做的只能是将有限信号做周期性的延拓，假定信号是以已有这段信号周期性无线重复的，这样我们就能分析了，实际上这给信号分析带来的影响并不是很大，有兴趣的同学可以查阅离散傅里叶分析相关理论知识。

对数字信号，我们使用更多的是快速傅里叶变换，即通常所说的 FFT。FFT 的原理这里我们就不做介绍了，有兴趣的同学请查阅数字信号处理相关资料（如 https://zhuanlan.zhihu.com/p/31584464；https://zh.wikipedia.org/wiki/快速傅里叶变换）。在 Matlab 中，FFT 函数的调用格式为

Y = fft(X)

Y = fft(X,n)

Y = fft(X,n,dim)

其中，X 为待分解的信号矩阵，可以是一个信号或一组长度相同的信号，dim 就表示需要分解的维度，是做行分解还是列分解。n 表示做 n 点 FFT，默认长度与信号长度相等，FFT 算法要求信号长度为 2^N，所以信号长度不为 2^N 时，将被自动补齐。返回值 Y 即分解出的各分量幅值和相位，用复数表示。需要稍加说明的是，FFT 函数返回的结果为双侧频谱结果，即频率范围为（-Fs,Fs）（Fs 为信号采样率），单我们知道负频率是没有物理意义的，实际上频率 f 分量的幅值为 2*abs(Y(f))，相位为 angle (Y(f))，直流(f=0)分量不需要乘以 2。若要直接得到单侧傅里叶分解结果，可以使用 fftshift()。这里举一个 help 中的案例，更多案例可参考 matlab help 中的 fft 函数。如图 3.2（左）中的信号，为两个不同频率

的正弦信号及噪声信号叠加得到的，看似杂乱无规律，但再频谱下课清楚看到两个正弦信号频率，如图 3.2（右）所示。

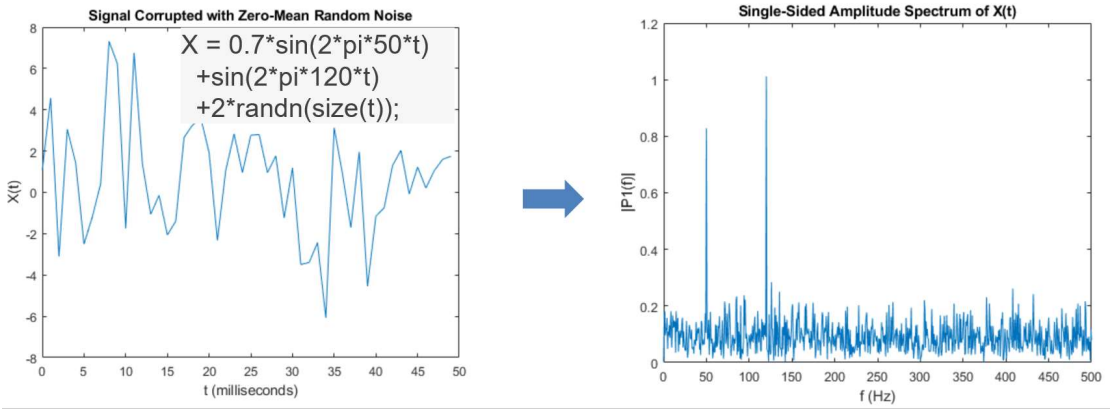

图 3.2 信号时域波形及频谱特征

案例程序：

```
% generate signal
Fs = 1000;            % Sampling frequency
T = 1/Fs;             % Sampling period
L = 1500;             % Length of signal
t = (0:L-1)*T;        % Time vector
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
X = S + 2*randn(size(t));
plot(1000*t(1:50),X(1:50));
title('Signal Corrupted with Zero-Mean Random Noise');
xlabel('t (milliseconds)');ylabel('X(t)');
% FFT
Y = fft(X);
P2 = abs(Y/L);          % 求幅值
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;
plot(f,P1);
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)');ylabel('|P1(f)|');
```

3.1.3 短时傅里叶分析

我们知道，傅里叶分解是对整段信号做分解，反映出来的信号频谱必然是全时间区间里的频谱，但一段实际信号经常是随时间变化的，前段时间信号的特征和后段信号的特征往往不同，如何才能反映频率随时间变化的问题呢，那就可以做短时傅里叶分解。顾名思义，短时傅里叶分解就是取一段时间的信号对其进行分解，移动时间窗口就可以分析整段时间的频谱了。

Matlab 中的短时傅里叶分解函数为 spectrogram()，调用格式为

s = spectrogram(x)

s = spectrogram(x,window)

s = spectrogram(x,window,noverlap)

s = spectrogram(x,window,noverlap,nfft)

其中，x 为待分解信号，window 为采用的窗函数，窗函数一方面是用于从原始信号中取出一段信号，另一方面是将窗口两端的信号弱化，以减小由于窗口长度选择人为引入的频率影响，有多种窗函数可以选择，常用的有 Hamming 窗等。noverlap 是两个相邻时间窗之间重叠的长度，nfft 为计算 FFT 的点数。

如图 3.3 所示，短时傅里叶分解实际上是将原信号分解成若干段，然后分别对每一段进行傅里叶分解的过程。值得注意的是，很显然窗口长度选得不同时所得结果也会右差别，即便对于周期信号，如果窗口长度不为整数个周期，也会引入额外的频率分量，为了减小窗口长度的影响，一般引入窗口函数，如图中红色线所示，用窗口函数与取得的函数相乘，然后再做分解。窗口函数有很多种可供选择，各有特点（参考 help 中 window functions）。另外，如果将整个时间段分成 N 段，matlab 会做 2N-1 次分解，即在两个窗中间也会进行一次分解，提高时间分辨率。

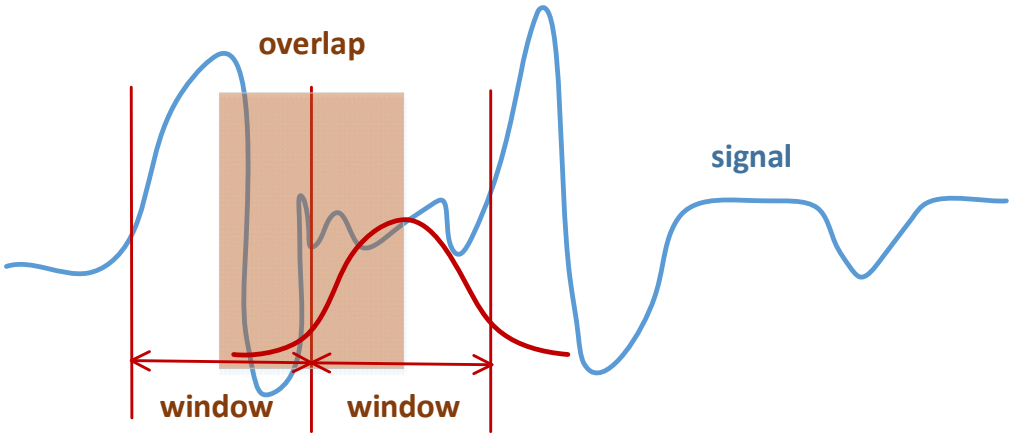

图 3.3 短时傅里叶分解的原理

短时傅里叶分解的案例见函数 spectrogram()的 help 文档。

3.2 信号频谱分析之小波分解

3.2.1 问题来源

我们在做项目时经常需要采集信号，由于环境的复杂性，采集到的信号中往往包含很强的噪声，我们经常会苦于如何很好地去除噪声。我们首先就会想到滤波，只要噪声和有用信号频率相差较大，便能将噪声滤除。这在理论上是没问题的，我们很多时候也会这么做。但我们也会遇到一些更麻烦的情况，比如有用信

号并不是一个简单的周期信号，而是频率在不断变化，这时滤波的难度将比较大。一种做法是进行短时傅里叶分析然后滤波，其实更加适合处理这类问题的是小波分解。

3.2.2 小波变换的原理

小波基函数是为了提高函数的频域分解的时间分辨率提出的。我们知道正弦函数是全域有值的，所以函数傅里叶分解出来的任何一个分量都是全域有值的，也就是说，某个分量在时间起始段具有的特性与结束段类似，这种特性就不适合提高时间分辨率。小波基函数则是一组局域基函数，它仅在一定区域内有值，这样可以大大提高时间分辨率。小波基函数以一个母函数为基础，其它基函数通过伸缩和平移得到，如母函数 ψ(t)，经过伸缩平移后可得

$$\psi_{a,b}(\mathrm{t})=\frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})$$

那么任意函数的小波变换为

$$W(a,b)=\frac{1}{\sqrt{a}}\int_{-\infty}^{+\infty}f(\mathrm{t})\cdot\psi(\frac{t-b}{a})dt$$

函数则可以通过逆变换进行重构还原

$$f(\mathrm{t})=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\frac{1}{a^2}W(a,b)\cdot\frac{1}{\sqrt{|\mathrm{a}|}}\psi_1(\frac{t-b}{a})dadb$$

式中 $\psi_1(t)$为 ψ(t)的对偶函数。

显然我们需要取不同的 a 和 b，以得到不同分量的系数（坐标），严格地说我们需要取遍所有的 a 和 b。可见小波变换与傅里叶变换原理上很接近，只是取不同的基而已。

小波分解的基有很多种取法，也有不少讲究，相关的论述需要参考专业书籍和文献，这里不讲。这里介绍几种典型的母小波基函数，如图 3.4 所示，可见它们形状类似正弦，只是仅作用于一段自变量区间，而非全局函数。在实际中选取哪一种小波函数则需要查看更加专业的资料。

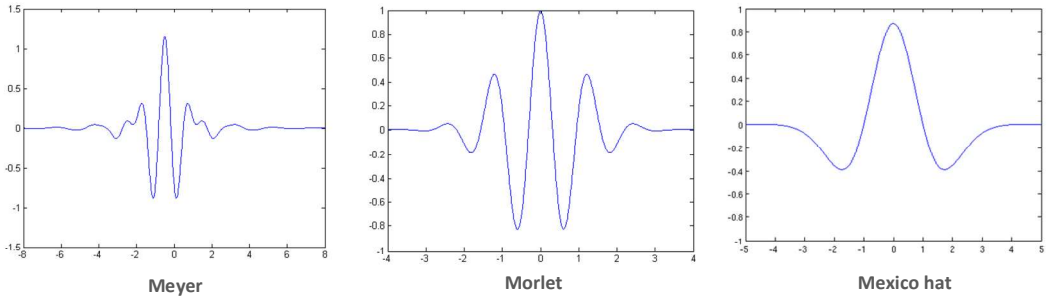

图 3.4 几种典型的母小波基函数

3.2.3 小波变换与傅里叶变换的比较

首先我们对傅里叶分解基函数（三角函数）和小波基函数（以 Morlet 函数为例）进行比较。从时域波形上看，两者都是正负振荡的，只是三角函数在全时间轴都有值，是无限延伸的，而 Morlet 函数却仅在一定区域内有值，且两端是平滑衰减的。根据这个特性，我们如果根据公式进行分解，显然 f(t)中仅有一段时间的波形对积分有贡献，这就是小波分解具有时间分辨能力的原因。

我们再对两种函数进行傅里叶变换，得到两者的频谱特性如图 3.5 所示，可以看出，三角函数的具有非常明确的选频特性，除频率 ω0 外的信号都将被滤掉，而小波函数滤波特性中同样有中心频率 ω0，但不同的是，它有一定的展宽，也正是这个频谱上的展宽，换取了时域上的分辨能力。

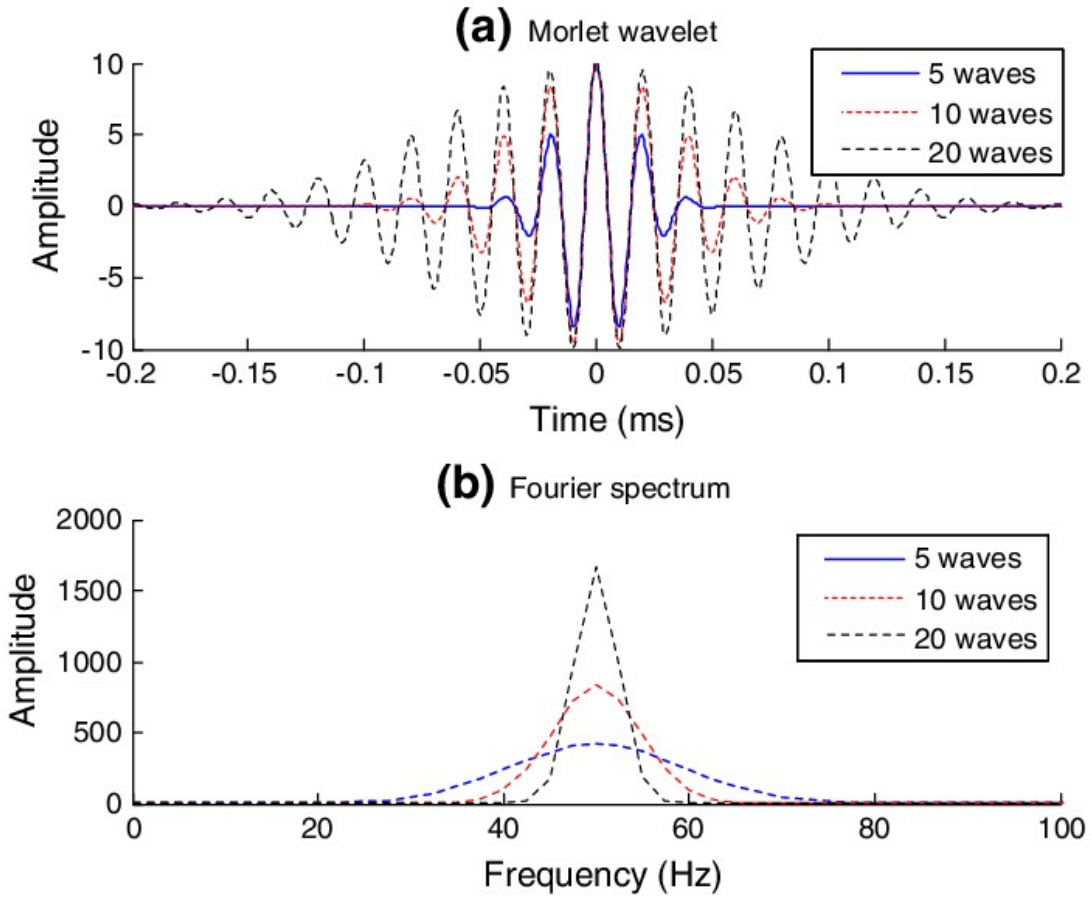

图 3.5 Morlet 函数及其频谱

从小波分解公式可以看出，小波分解的过程实际上可以看作是对函数 f(t)向小波基取投影的过程，也是求某一个基函数对应的坐标的过程。对比傅里叶分解，可以看出两者是完全类似的，从傅里叶分解公式可以看出，公式实际上可以看作是 f(t)与正弦函数（基）的内积运算（投影运算，或求坐标的运算），分解过程实际上是求函数 f(t)向不同频率的正弦函数求投影的过程，如图 3.6 所示。傅里叶分解过程还可以看作是滤波过程，即将 f(t)中不同频率分量用正弦函数将其滤出来

（因为我们知道，不同频率的正弦函数相互正交，即乘积的积分为 0）。小波分解也是类似的，分解过程如图 3.7 所示，可看作是采用多个不同尺度（不同频率和相位）的基函数作为窗函数，对待分解函数不断进行短时傅里叶分解的过程。

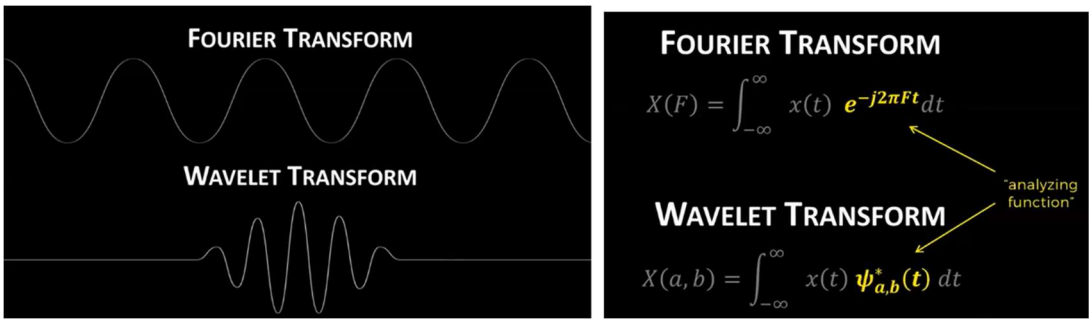

图 3.6 傅里叶分解与小波分解的比较

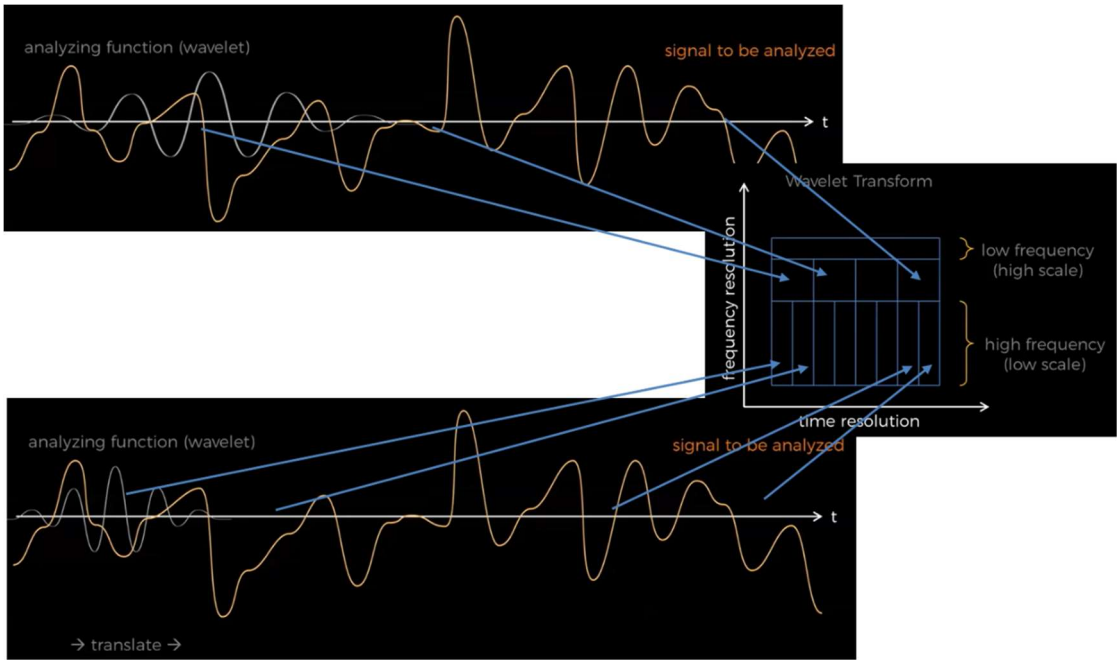

图 3.7 小波分解过程

3.2.4 连续小波变换

前面介绍了小波变换的基本原理和分解流程，回想一下，要做小波变换，我们需要一些什么输入量，分解结果又是什么呢？输入量首先是一个信号波形数据，其次是小波基函数类型，分解结果每个时刻不同频率分量的大小。在 Matlab 中有对应的连续小波变换函数 cwt()，其基本调用格式如下：

```
wt = cwt(x)
wt = cwt(x,wname)
[wt,f] = cwt(__,fs)
```

其中，x 为信号波形数据，wname 为小波基名称，matlab 中包含很多种，不同

小波在滤波特性上具有一定差别，详细信息可通过查阅函数 wfilters 获取。wt 为变换结果，为二维矩阵，每一行代表一个频率尺度，为这个频率分量随时间演化的数据。如图 3.8 为某次地震信号，右边为运行 cwt 函数后的结果，从小波分析结果上很容易看出信号的频率演化信息。

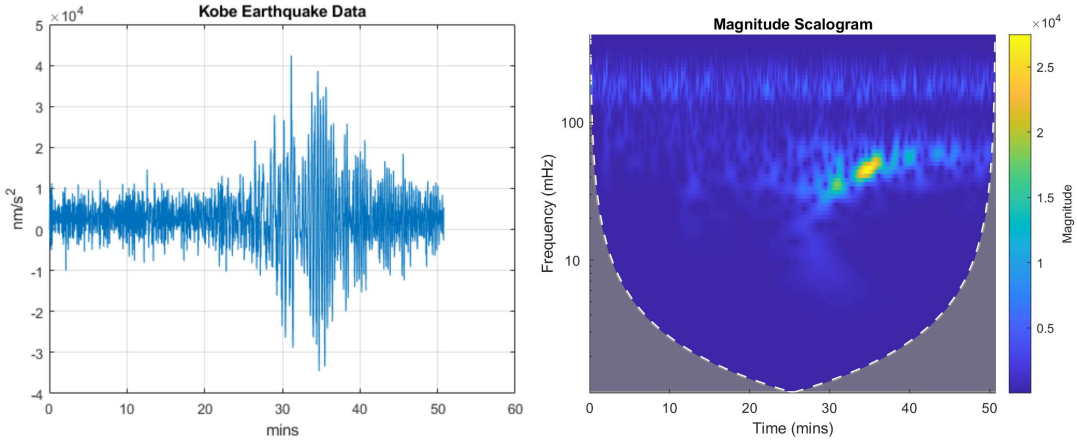

图 3.8 某次地震信号及其小波分解结果

3.2.5 离散小波变换

在计算机中，我们总是处理数字信号，尽管前面的 cwt 函数也是处理的数字信号，但它是将连续分解公式数字化得到的。我们实际应用中有时候还需要进行快速分解，就如同快速傅里叶分解 FFT 一样，我们也需要快速小波分解。

这里的相关理论需要数字信后处理信号的知识，鉴于大家还没有学习，这里尽量少的涉及相关的理论。我们先给出典型快速小波变换的流程，如图 3.9 所示。g[n]和 h[n]为一对小波滤波器，分别为低通和高通滤波器。g[n]是低通滤波器的冲激响应，g[n]表示卷积运算，实际上是实现滤波器功能，稍后解释卷积与滤波之间的关系。↓2 表示下采样，即间隔 1 个点取 1 个值出来。低通滤波的输出为近似（轮廓，approximation）分量，高通滤波的输出为细节（detail）分量。按流程图 3.9，进行一次分解将得到高频分量和剩余低频分量，下采样的结果是信号长度缩小为一半，在下次再与滤波器函数做卷积时等效为将滤波器函数做一次拉伸，即滤波器截止频率减半。不断重复下去，就能把高频细节逐渐过滤出来，同时保留低频近似（轮廓）分量。

Matlab 中进行单层快速小波分解的函数为 dwt()，调用格式如下：

[cA,cD] = dwt(x,wname)

[cA,cD] = dwt(x,LoD,HiD)

下位待分解信号，wname 为选取的小波函数名，LoD 和 HiD 为一组分解滤波器，分别实现低通滤波和高通滤波，对应图 3.9 中的 h[n]和 g[n]，通常采用函数 wfilters()构造。

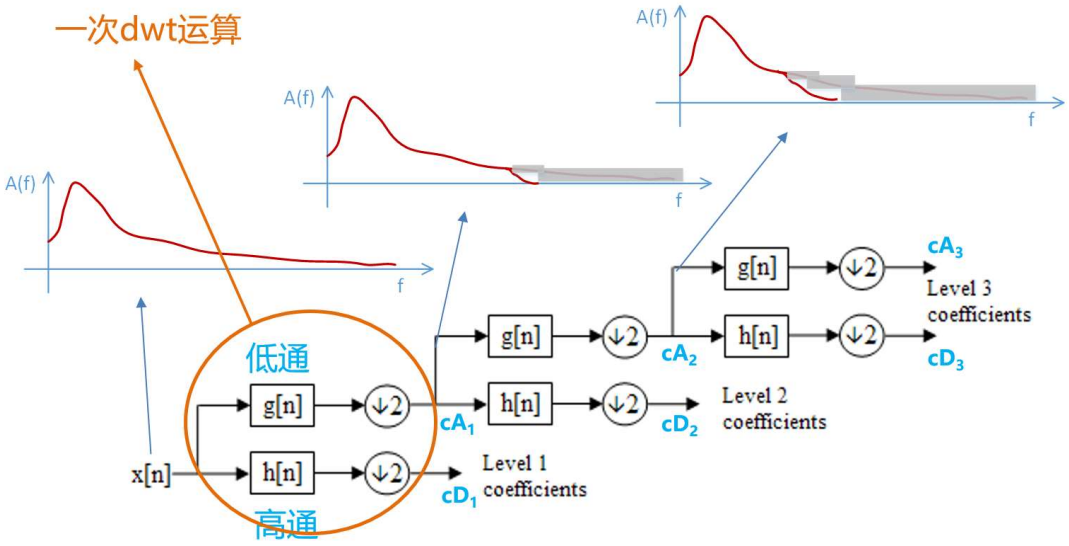

图 3.9 快速小波分解过程

下面通过一个案例讲解 dwt()函数的原理和用法。下面一段程序完成一个信号的小波分解，并在频域中删除一些分量后进行信号重建，也就是滤波过程。

```
tt=0:5e-6:(2^12-1)*5e-6;
freq=[200*ones(size(tt(1:floor(2*length(tt)/3)))),2000*ones(size(tt(1+floor(2*length(tt)/3):length(tt))))];
signal_0=2*sin(2*pi*freq.*tt);
noise=rand(size(signal_0));
signal=signal_0+noise;
plot(tt,signal);
% wavelet by db2 filter
wname = 'db2';
[LoD,HiD,LoR,HiR] = wfilters(wname);
[cA1,cD1]=dwt(signal,LoD,HiD,'mode','per');
[cA2,cD2]=dwt(cA1,LoD,HiD,'mode','per');
[cA3,cD3]=dwt(cA2,LoD,HiD,'mode','per');
[cA4,cD4]=dwt(cA3,LoD,HiD,'mode','per');
tt1=dyaddown(tt);
tt2=dyaddown(tt1);
tt3=dyaddown(tt2);
tt4=dyaddown(tt3);
% observe cA1~cA4, cD1~cD4
plot(tt,signal,'b');hold on;
plot(tt1,cA1(1:length(tt1)),'r');plot(tt2,cA2(1:length(tt2)),'m');
plot(tt3,cA3(1:length(tt3)),'g');plot(tt4,cA4(1:length(tt4)),'c');
hold off;
plot(tt1,cD1(1:length(tt1)),'r');hold on;plot(tt2,cD2(1:length(tt2)),'m');
plot(tt3,cD3(1:length(tt3)),'g');plot(tt4,cD4(1:length(tt4)),'c');
hold off;
```

```
% idwt 信号重建
% let cD1, cD2,cD3 =0
cA3_R=idwt(cA4,cD4,LoR,HiR,'mode','per');
cA2_R_3=idwt(cA3_R,zeros(size(cD3)),LoR,HiR,'mode','per');
cA1_R_3=idwt(cA2_R_3,zeros(size(cD2)),LoR,HiR,'mode','per');
signal_rec_3=idwt(cA1_R_3,zeros(size(cD1)),LoR,HiR,'mode','per');
% let cD1, cD2 =0
cA2_R_2=idwt(cA3_R,cD3,LoR,HiR,'mode','per');
cA1_R_2=idwt(cA2_R_2,zeros(size(cD2)),LoR,HiR,'mode','per');
signal_rec_2=idwt(cA1_R_2,zeros(size(cD1)),LoR,HiR,'mode','per');
% let cD1 =0
cA1_R_1=idwt(cA2_R_2,cD2,LoR,HiR,'mode','per');
signal_rec_1=idwt(cA1_R_1,zeros(size(cD1)),LoR,HiR,'mode','per');
% complete idwt
signal_rec=idwt(cA1_R_1,cD1,LoR,HiR,'mode','per');
```

所构造信号波形如图 3.10 中 signal，HiR 为高通重建函数。LoD 与 HiD 构成一对用于小波分解，LoR 和 HiR 为一对，用于信号的重建（用 wfilters()可一次完成构造）。dwt 为离散小波分解函数，相当于执行一次滤波和一次下采样。cA1~cA4 分别为执行 4 层小波分解得到的低频分量，cD1~cD4 分别为执行 4 层分解后得到的高频细节分量。高频分量和低频分量的结果在这里并没有画出，读者可以自行绘画。可以看出，随着分解层次的提高，高频细节 cD 中逐渐滤除更多的低频分量，相应地，cA 中逐渐丢失高频分量。

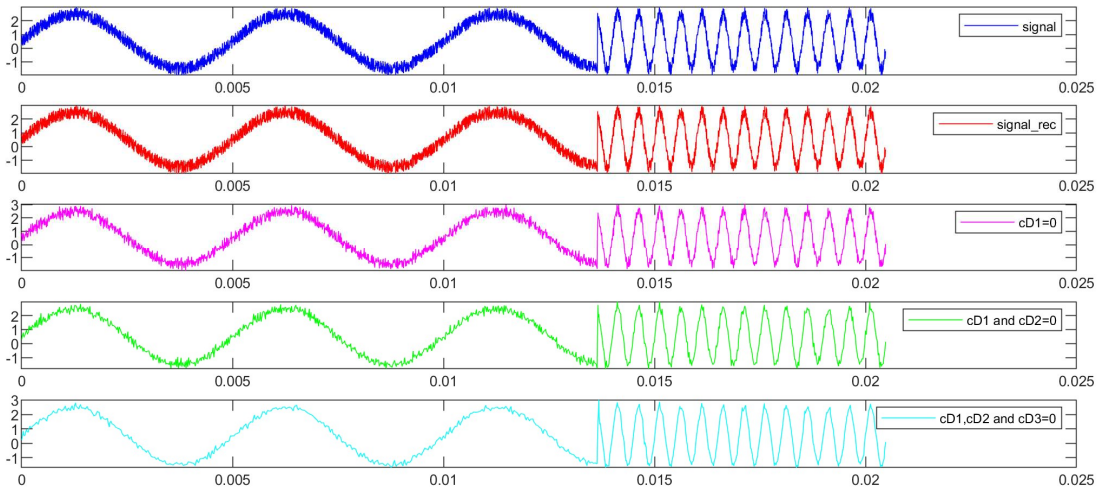

图 3.10 信号及重建结果

Matlab’中进行小波分解还可以采用 wavedec()函数可实现，它可一次性完成多层小波分解，信号的重建可用 waverec()实现，调用格式如下：

```
[c,l] = wavedec(x,n,wname)
```

```
[c,l] = wavedec(x,n,LoD,HiD)
x = waverec(c,l,wname)
x = waverec(c,l,LoR,HiR)
```

下面是一段示例代码：

```
level=4;
[c,l]=wavedec(signal,level,LoD,HiD);
approx = appcoef(c,l,'db2');       % ca4
[cd1,cd2,cd3,cd4] = detcoef(c,l,[1 2 3 4]);
cd1_1=zeros(size(cd1));cd2_1=zeros(size(cd2));
c_1=[approx,cd4,cd3,cd2_1,cd1_1];
signal_rec_waverec=waverec(c_1,l,LoR,HiR);
figure();
subplot(2,1,1);
plot(tt,signal,'b');legend('signal');
subplot(2,1,2);
plot(tt,signal_rec_waverec,'r');legend('signal\_rec when cD1 and cD2=0');
```

多层小波分解函数 wavedec()返回的变量 c 和 l 如图 3.11 所示。其中用到了与 wavedec 和 waverec 配套使用的函数有 appcoef()和 detcoef()，分别用于获取低频和高频分量。我们可以将与噪声相关的分量置零或乘上某个系数，甚至只将某个分量的其中一段时间置零，以达到最好的滤波效果。

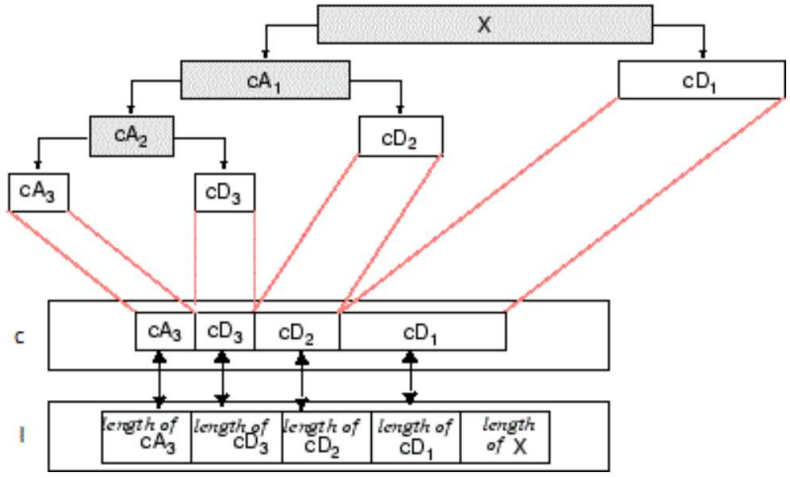

图 3.11 wavedec 函数返回参数的意义

实际上，在小波分解工具箱中，还有一个比较智能的函数 wdenoise()，可以直接根据小波分解结果进行降噪，读者可自行测试。

小波分解在工程实际中具有十分广泛的应用，如信号去噪、图像压缩、图像特征提取等等，想想傅里叶变换的广泛用途便知。

3.2.6 练习：音频信号滤波

音频文件见 QQ 群” dylanf_noise.m4a”。尝试分析信号的频谱特征，滤除音频中的噪声，比较去除部分频率分量后的效果。

3.3 图像处理

3.3.1 基础知识

Matlab 中的图片类型主要有真彩图、灰度图、二值图、索引图几种，简单介绍如下。真彩图（RGB 图，true color 图）采用一个 M*N*3 的矩阵存放图像信息，M 和 N 分别为水平和竖直方向的像素数，(:,:,1)、(:,:,2)和(:,:,3)分别代表各个像素的 R、G 和 B 值。RGB 值默认情况下用 8 位整数表示，也可以转换为 double 等格式，特别是在对图像做一些运算处理时需要进行转换，以获得比较高的计算精度。灰度图就是我们常说的黑白图，采用 M*N 的矩阵存放图像信息。二值图则是仅包含黑和白两种颜色的图，同样为 M*N 的矩阵，但每个像素的值仅有 0 和 1 两种可能，是逻辑数据类型。索引图借助一个调色板，将特定数目的 RGB 值存放在调色板中，图中某一点的值则是一个索引 n，指向 RGB 调色板中某一行，所以真正某一像素的 RGB 值为调色板中第 n 行的 RGB 值。

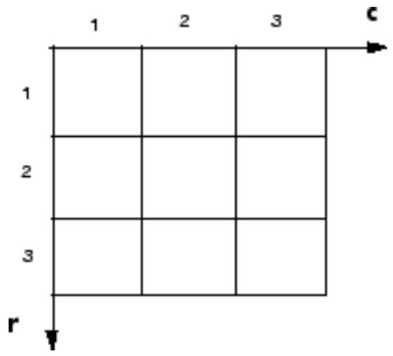

图 3.12 图像坐标

不同类型图像之间可以进行转换，常用转换函数有 rgb2gray()、rgb2ind()、gray2ind()、ind2gray()、ind2rgb()、imbinarize()等。各转换函数的用法可参考各函数的 help 文档，这里仅举其中一个例子说明 imbinarize 的用法。如图 3.13 上为一个页面的扫描图，运行以下代码后变成下图所示效果。

```
BW = imbinarize(I,'adaptive','ForegroundPolarity','dark','Sensitivity',0.4);
```

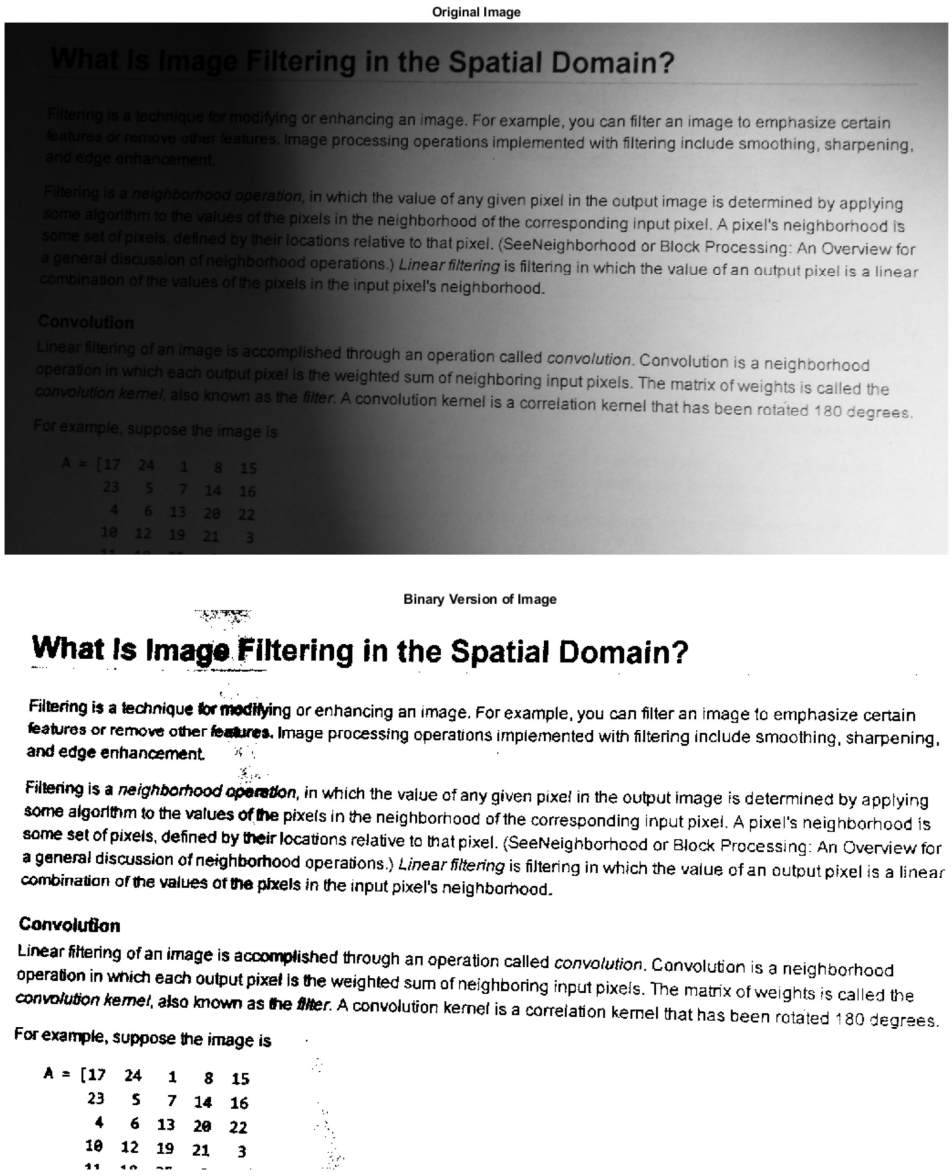

图 3.13 二值化前后的效果比较

实际上，二值图尽管信息量相对较小，但却是大量图像处理函数的基础，利用二值图及其处理结果，与原图进行合适运算，可以实现图像的很层次处理。

Matlab 中的颜色空间也有多种，如 RGB、HSV、YIQ、YCbCr 等，不同颜色空间间可进行转换，这里给出 HSV 和 RGB 的对比，如图 3.14 所示。

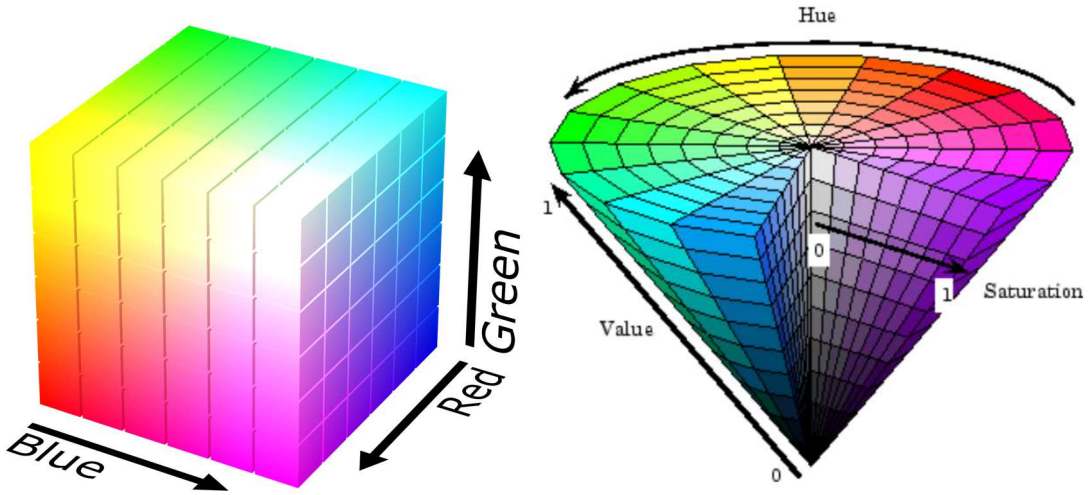

图 3.14 RGB 和 HSV 颜空间的对比

3.3.2 图像的读写

图像的读写函数主要有 imread()，imshow()，imwrite()，imfinfo()等，分别为读图、显示图、写图和获取图像信息函数。

对比度是图像的一项基本质量指标，在 matlab 中可对其进行一定调节。我们先需要知道灰度直方图，它是一项统计指标，即统计图中不同灰度值的像素个数，并化成直方图的形式，查看直方图的函数为 imhist()。如图 3.15（左）为一幅对比度较低的图，其灰度直方图如右图。

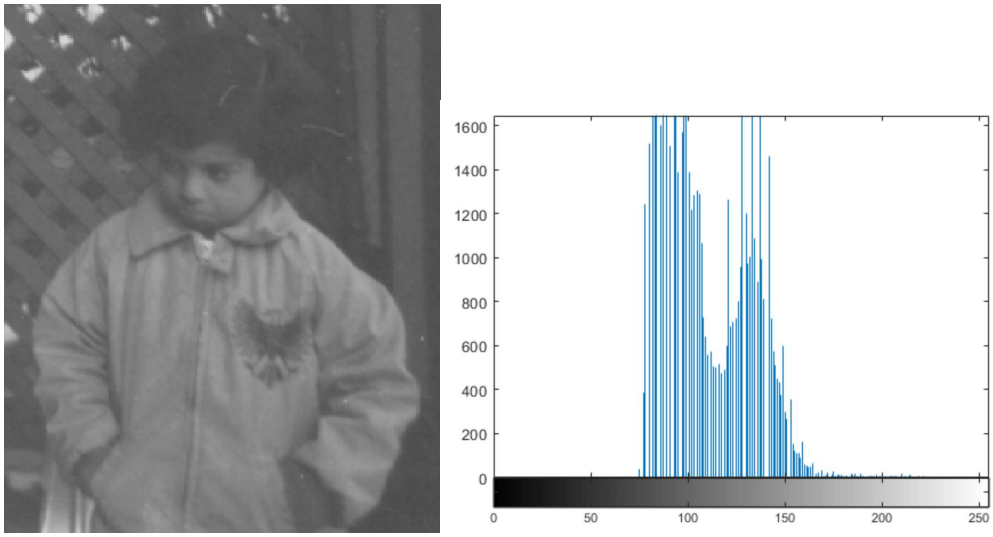

图 3.15 低对比度图及其灰度直方图

对该图执行一次 histeq()命令，对灰度进行一次均衡处理过后，结果变成图 3.16，可以看出，图像对比度提高了，同时其灰度值范围在原来基础上增大了不少。这实际上是将原图中灰度值进行扩充使其基本布满整个灰度空间的操作。

对图像对比度进行调节的函数还有 imadjust()，读者可自行查看 help 文档。

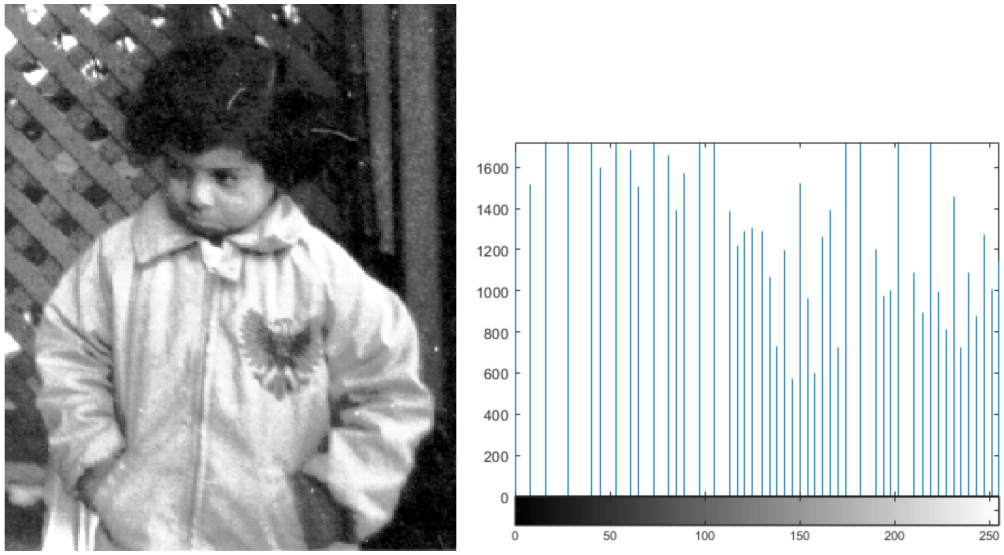

图 3.16 对比度提高过后的效果及其灰度直方图

3.3.3 图像分割

图像分割是高级图像处理的基础，是机器人视觉、医学自动诊疗等的基础，如车牌识别和读取、小球的抓取、磁共振图中病灶分割等。涉及到的处理方法很多，详细的介绍超出了本课程内容，这里仅做一个初级的介绍。

直接在空间坐标上进行图像分割的方法主要基于图像边沿的灰度值较大变化提取边沿图像，然后对其进行膨胀、腐蚀、填充等操作进行修理，获得目标区块的准确边界，再与原图进行一定运算实现分割。实际图像的详细分割处理过程各部相同，方法也比较多，这里以 help 中的一个简单例子进行介绍。

我们尝试分割一个图片中的硬币，如图 3.17。如果我们能将每个硬币提取出来，然后跟标准硬币做对比，就可能能识别出图中硬币的币值，这里我们仅做到分割这一步。

图 3.17 待分割的硬币图片

首先观察这个图片，整体上来说比较清晰，硬币边沿也比较清楚，所以可以转成灰度图，然后尝试二值化，执行以下代码：

```
im_coin_bw=imbinarize(im_coin_gray,"adaptive","Sensitivity",0.55);
```

```
im_comp_fill=imfill(im_coin_bw,'holes');
figure();imshowpair(im_coin_bw,im_comp_fill,'montage');
```

可得图 3.18 效果。可见，二值化后硬币轮廓已经很清晰，我们可以使用 imfill()函数将硬币内部填充起来，就已经比较好地获得了硬币的轮廓。

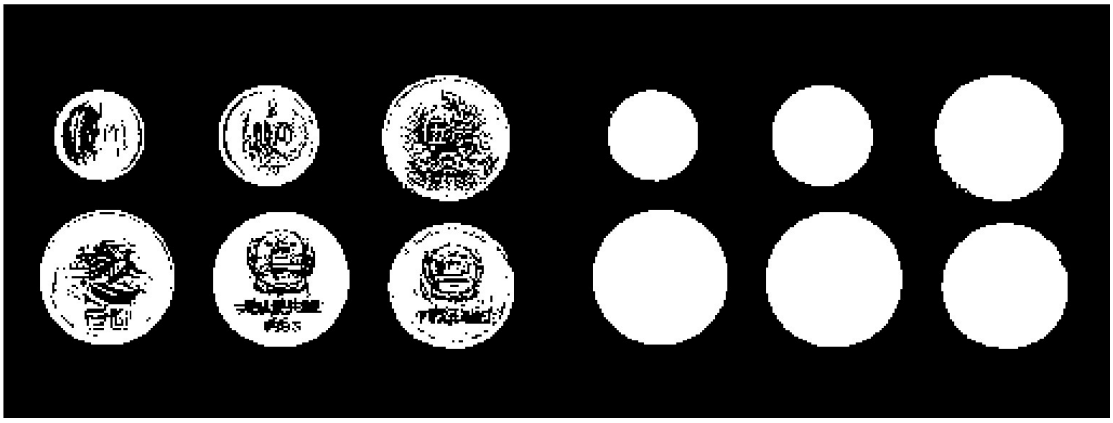

图 3.18 灰度图二值化后的效果和填充后的结果

我们采用边界检测方法试试。边界检测函数为 edge()，它是根据图像的微分进行检测的，得到图 3.19 左图所示效果，同样可以使用 imfill()填充。当然我们也可以尝试形态操作函数，如 imdilate()和 imerode()等函数。

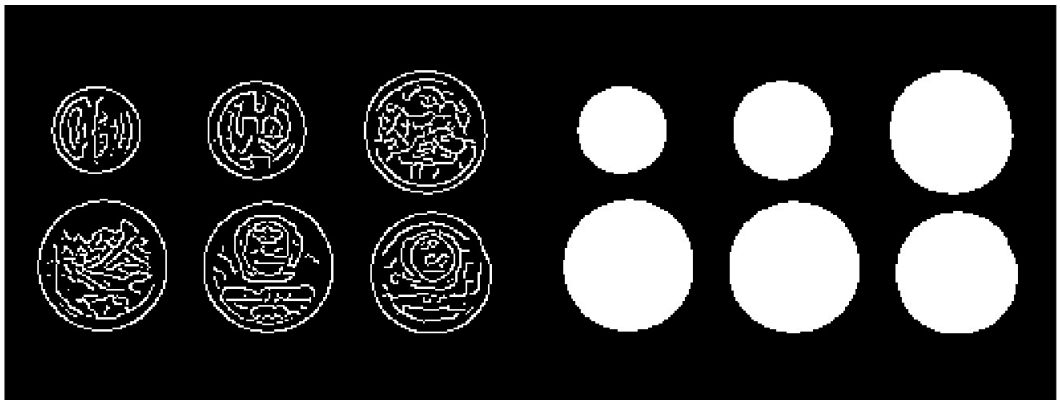

图 3.19 edge()函数执行效果

此外，对圆形目标，我们还可以直接使用 imfindcircle()搜索图中的硬币，如下代码可获得图 3.20 的效果。

```
[centers1,radii1]=imfindcircles(im_coin_gray,[10…
    150],'ObjectPolarity','bright',"Sensitivity",0.9);
viscircles(centers1, radii1,'Color','b');
```

图 3.20 imfindcircle()执行后画出的圆

在实现硬币区域的获取后，我们可以对图中每个硬币所处区域进行标记，使用如下代码：

```
s=regionprops(im_edge_fill,'Centroid');
area=regionprops(im_edge_fill,'Area');
L=bwlabel(im_edge_fill);
j1=1;
for i1=1:length(area)
    if area(i1).Area>1200
        ind_centr(j1)=i1;
        x_pos_coin(j1)=s(i1).Centroid(1);
        y_pos_coin(j1)=s(i1).Centroid(2);
        j1=j1+1;
    end
end
area_coin=area(ind_centr).Area;
imshow(im_edge_fill);hold on;
plot(x_pos_coin,y_pos_coin,'*');
```

这样可以实现硬币区域的分割。

3.3.4 练习：车牌识别