# 信号与系统－ Matlab 综合实验[1]

谷源涛　　应启衍　　郑君里

二〇〇七年八月十四日

[1]草稿

# 目录

# 第一章　绪论

## 第一节　信号与系统

“信号与系统”是电子工程与信息类专业的一门重要的专业基础课，也是国内各院校相应专业的主干课程。因而加强“信号与系统”的课程建设工作在电子信息类专业人才的培养过程中起着至关重要的作用。

加强实践教育是高等教育教学改革的重要组成部分，是培养创新性人才的重要环节。多年来，如何开设和加强“信号与系统”课的实践环节一直是广大任课教师关注的问题，“信号与系统”课程有很多基本概念，在这些概念的背后，又有广泛的应用背景。能够通过可视化、互动的计算机仿真工具说明这些概念，让学生深刻理解这些概念，通过系统仿真了解其应用，对学好课程培养创造性理念和实践能力非常必要。

---

### Matlab 知识－简介

Matlab 是一门计算机程序语言，取名源于 Matrix Laboratory，意在以矩阵方式处理数据。Matlab 将数值计算和可视化环境集于一体，并提供很多函数，功能强大且直观简便，因而应用范围非常广泛。一般认为 Matlab 的典型应用包括：(1) 数值计算与分析；(2) 符号运算；(3) 建模与仿真；(4) 数据可视化；(5) 图形处理及可视化；(6) 基于图形用户界面的应用程序开发。这些内容在本书中均有涉及。

Matlab 由主包和数十个可选的工具箱组成。主包带有功能丰富和完备的数学函数库，大量复杂的数学运算和分析都可以直接调用 Matlab 函数求解。工具箱为特定的学科和研究领域提供丰富的处理工具支持，本书主要涉及和“信号与系统”相关的若干工具箱，包括信号处理工具箱 (Signal Processing Toolbox)、通信工具箱 (Communication Toolbox)、控制系统工具箱 (Control System Toolbox) 等。

Matlab 启动后界面如图 1.1 所示。下面介绍其主要组成部分，通过这些介绍希望同学们能够基本掌握 Matlab 的使用方法。

**命令窗口 (Command Window)** Matlab 最基本的窗口，用户通过它来调用 Matlab 编译/解释器。正常情况下提示符为“>>”，表示 Matlab 进入准备工作状态。用户可在提示符后输入运算指令和函数调用等命令，Matlab 将迅速显示出结果并且再次进入准备工作状态。注意：如果命令后带有“;”，Matlab 执行命令但不显示结果。进入调试状态后，提示符将变成“K>>”。如果按上下键，Matlab 会按顺序依次显示输入命令的历史记录。

**命令历史记录 (Command History)** 这里记录有用户在“命令窗口” 中输入过的命令，以及每次启动 Matlab 的时间等信息。如果双击某条历史记录，则 Matlab 会再次执行该命令。

**工作空间 (Workspace)** 显示当前计算机内存中 Matlab 变量的名称、数学结构、字节数及其类型等。用户可以双击查看并修改该变量的内容，也可以将某变量存储到文件中或者从文件中载入某变量。

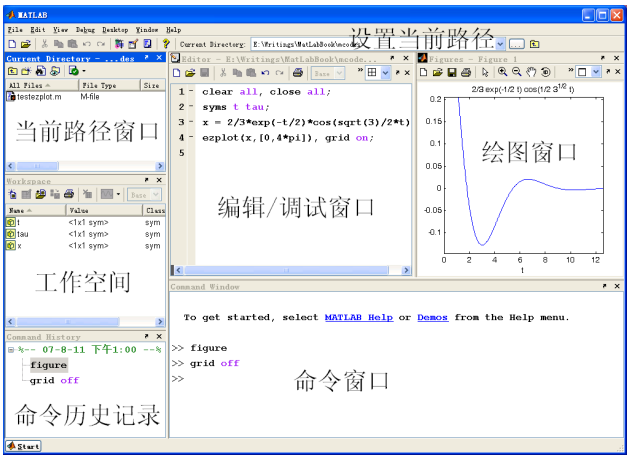


图 1.1: Matlab 7.0 用户界面

**编辑/调试窗口 (Editor)** 简单的运算可以在命令窗口中直接输入；但若要执行复杂的计算，最好在文件中全部编辑好运算指令再统一执行，编辑/调试窗口就是 Matlab 为此提供一个集成环境。该环境除了支持基本的编辑功能外，还提供解释执行程序的调试功能，例如“设置断点 (F12) ”和“单步执行 (F10) ”等。 Matlab 程序文件扩展名为“.m”，以文本文件形式保存。有两种方式运行程序文件：一是在命令窗口输入文件名；二是在编辑/调试窗口选择“Debug→Run”或按“F5”键运行程序。注意：对于不是通过 Matlab 主界面打开的“.m”文件（比如在 Windows 资源管理器中双击打开），编辑/调试窗口不支持调试功能。

**图形窗口 (Figure)** 数据可视化是 Matlab 强大功能的重要体现，用户编程绘制图形时会显示在图形窗口中。用户还可以编程或者手工修改各个图形元素的颜色、线型等属性。

**设置当前路径 (Current Directory)** 用于选择当前工作路径。工作路径下的文件，是可以通过在命令窗口中输入文件名的方式直接调用的（只要在 Matlab 的搜索路径列表中，都可以通过这种方式调用，搜索路径的概念后面再做详细介绍）。如果试图在编辑/调试窗口中运行一个非当前工作路径下的文件， Matlab 会提示你是否要更改当前路径。

**当前路径窗口 (Current Directory)** 显示当前工作路径内的所有文件，用户可以在这里创建或删除一个文件，也可以双击一个文件在“编辑/调试窗口”中打开。

除上述窗口外，常见的 Matlab 界面还包括“帮助窗口”、“ Simulink Library Browser 窗口”、“ Simulink Model Editor 窗口”、“ GUIDE 窗口”、“Profile窗口”、“Array Editor” 等，本书后面在需要的地方会对这些界面做详细介绍。

学习 Matlab 的最详细教材是它的帮助文件。*请务必安装完整的帮助文档。* 获取帮助信息有两种方法：一是直接在命令窗口输入“help 函数名或命令”；二是在帮助窗口中浏览或搜索相应信息。参考 Matlab 的 Demo 程序也是学习编程的重要途径。

## 第二节　信号的描述

“信号与系统”中描述信号的基本方法是写出它的数学表达式，这对应于 Matlab 中的符号表示法，我们将随后介绍。在 Matlab 中，描述信号的基本方法是数值表示法。此方法中连续时间信号和离散时间信号的界限已经消失，统一以采样信号的形式用矢量表示，采样间隔越小，信号连续性越强。同理，矩阵可以表示多个同样长度的信号。Matlab 对信号的描述正是通过对矢量和矩阵的定义和操作实现的。

**例 1.1** 用数值方法表示离散信号 $x(n) = 2nu(n)$ 的前六个非零值和连续信号 $y(t) = \sin(2\pi t)$（$0 \le t \le 1$）。

**解** 对于离散信号 $x(n)$，题意要求定义下标 $1 \le n \le 6$ 的样值。因而在命令窗口中输入

```
>> n = [1:6];
>> x = 2*n
```

将看到 Matlab 回显的 x 值

```
x =
 2 4 6 8 10 12
```

对于连续信号 $y(t)$，我们定义其采样间隔为 0.1，即

```
>> t = [0:0.1:1];
>> y = sin(2*pi*t)
```

Matlab 回显的 y 值为

```
y =
Columns 1 through 9
 0 0.5878 0.9511 0.9511 0.5878 0.0000 -0.5878 -0.9511 -0.9511
Columns 10 through 11
 -0.5878 -0.0000
```

例 1.1 中 pi 是 matlab 自定义的常量，代表圆周率 $\pi$。

---

### Matlab 知识－矢量和矩阵

Matlab 中的“:”用于构造一个元素为等差数量的行矢量，如例 1.1 中“0:0.1:1”表示该行矢量第一个元素是“0”，第二个是“0.1”，然后依次增长到“1”共 11 个元素；如果忽略中间项则意味着增量为“1”，如例 1.1 中的“1:6”定义了从 1 到 6 共六个整数组成的行矢量。

在 Matlab 中用方括号“[ ]”定义矩阵（矢量是矩阵的特例），方括号内逗号“,”或空格分隔矩阵的列，分号“;”或回车键“Enter”分隔矩阵的行。例如：

```
>> A=[a11 a12 a13;a21 a22 a23]
```

定义了一个 $2 \times 3$ 阶矩阵 A，其中每个元素 aij 可以为数值或变量，如是变量必须先赋值。用 A(i,j) 引用第 i 行第 j 列的元素。

---

最普遍的工程任务是对真实的采样数据进行处理，因而数值法描述信号是工程专业的基础，也是 Matlab 的传统优势。但从 5.3版 开始，Matlab 符号运算功能获得极大增强，到了 7.0 版，它已经号称和符号运算专用计算语言 Maple 和 Mathematic 等的处理能力相差无几。符号方法描述信号的优点是简单直观，理论性强。如下例。

**例 1.2** 用符号方法表示离散信号 $x(n)=2n$ 和连续信号 $y(t)=\sin(2\pi t)$ 。

**解** 我们对 x(n) 直接定义，输入

```
>> x=sym('2*n')
```

可看到 Matlab 回显出

```
x =
2*n
```

对于 y(t) 我们先定义出 t 再表示出 y(t)，输入

```
>> syms t y
>> y=sin(2*pi*t)
```

可看到 Matlab 回显出

```
y =
sin(2*pi*t)
```

下面我们想计算出 n 取值从 1 到 6 时的 x 值，可以用

```
>> x1=subs(x,'n',[1:6])
```

将回显

```
x1 =
 2 4 6 8 10 12
```

即 x1 中已经被赋予了数值。

---

**Matlab 知识－符号表达式基础**

在 Matlab 中，用 var=sym(str) 和 syms var1 var2 ··· 定义符号变量。符合表达式到数值变量的转换可以用 subs(f,x,y)，意为用 y 替换掉表达式 f 中的 x，如果 y 仍是符号变量，则 subs 实现符号替换。

---

## 第三节　信号的运算

信号的运算包括延时、反褶、压扩、微分、积分以及两信号的相加或相乘等。Matlab 中的信号运算方法也包括数值计算法和符号计算法，先介绍数值计算法。

---

**Matlab 知识－数学运算符**

Matlab 的数学运算符号，如表 1.1 所示。可见，Matlab 的运算以矩阵为基本单元，既支持矩阵之间的四则运算，又支持两个矩阵对应各个元素之间的运算，即各种带“.”的组合运算符号，称作数组运算。另外需要注意，左除的除数在左边，被除数在右边，即和该符号的倾斜方向表达的意思相同。

---

**例 1.3** 已知 $x(t)=\sin(2\pi t)u(t)$，$y(t)=\mathrm{e}^{-t}u(t)$。试计算 $t\in[0,2]$ 区间的 $z_1(t)=2x(t)$，$z_2(t)=x(t-1)$，$z_3(t)=x(2t)$，$z_4(t)=x(t)+y(t)$，$z_5(t)=x(t)y(t)$。

**解** 定义采样间隔 0.05 和 $x(t)$，$y(t)$，可直接计算得到 $z_1(t)$，$z_4(t)$ 和 $z_5(t)$

表 1.1: Matlab 中的数学运算符

| 名称 | 说明 | 名称 | 说明 |
|---|---|---|---|
| + - | 矩阵加，矩阵减 | * | 矩阵乘 |
| / \ | 矩阵右除，矩阵左除 | ^ | 矩阵求幂 |
| .* .^ | 数组乘，数组求幂 | ./ .\ | 数组右除，数组左除 |
| ' .' | 共轭转置，转置 | = | 赋值 |

```
>> t = [0:0.05:2];
>> x = sin(2*pi*t);
>> y = exp(-t);
>> z1 = 2*x;
>> z4 = x + y;
>> z5 = x.*y;
```

为了求 $x(t)$ 延时得到的 $z_2(t)$，定义一个全零的临时变量 temp，然后把 $x(t)$ 拼接到 $t < 1$ 的 temp 后面，即得到延时 1 的结果

```
>> temp = zeros(size(x));
>> z2 = [temp(t<1),x(t<=1)];
```

为了有足够多的数据供抽取，重新定义一段较长的 $x(t)$，然后从中隔一个采一个，即可得到 $x(2t)$，并且数据长度也满足要求

```
>> tL = [0:0.05:4];
>> xL = sin(2*pi*tL);
>> z3 = xL(1:2:length(xL));
```

例 1.3 中 exp 和 sin 一样，是 Matlab 提供的数学运算库函数；size 函数返回一个变量的大小；zeros 函数生成一个指定大小的全零矩阵。上例中较难理解的就是

```
>> z2 = [temp(t<=1),x(t<1)];
```

下面我们详细解释这句话执行的过程。首先执行的是比较运算 t<1，它的计算结果也是一个矢量，其中元素根据比较结果分别是 1（真）或 0（假）。再把这个比较结果作为矢量 temp 的下标，即 temp(t<1)，返回 t 中对应为真的这些下标的元素，即时间长度为 1 秒（从 0 秒到 1 秒）这么多的零值（用来作为延时 1 秒的填充物）。接下来再看 x(t<=1)，它的道理类似，即取了 x(t) 前 1 秒的数据，然后补在 1 秒的零值的后面，从而得到 2 秒的数据，满足了题意要求。

### Matlab 知识－比较和逻辑运算符

Matlab 中的真 (1) /假 (0) 和数值量完全相同并可参与运算，因而熟练运用比较运算和逻辑运算可以灵活巧妙的描述和操作信号。Matlab 提供的比较和逻辑运算符如表 1.2 所示，其中 all 和 any 是两个有特色的逻辑函数，请同学们课下充分练习。

表 1.2: Matlab 中的比较和逻辑运算符

| 名称 | 说明 | 名称 | 说明 |
|---|---|---|---|
| == | 等于 | ~= | 不等于 |
| > >= | 大于，大于等于 | < <= | 小于，小于等于 |
| & | 与 | \| | 或 |
| ~ | 非 | xor(a,b) | a和b做异或 |
| any(a) | a 中有元素非零则为真 | all(a) | a 中所有元素都非零则为真 |

## 第四节　绘制信号波形

数据可视化是 Matlab 的重要功能之一。例 1.3 中生成了多种信号，下面观察它们的波形特征。

**例 1.4** 对比绘制例 *1.3* 中各种信号的波形。

**解**

```
>> figure;                                % 生成新图框
>> hold on;                               % 进入绘图保留模式
>> plot(t,x);                             % 以默认格式绘制 t-x
>> plot(t,z1,'r');                        % 用红色实线绘制 t-z1
>> plot(t,z2,'k-o');                      % 用黑色实线带圆圈标记绘制 t-z2
>> plot(t,z3,'g--');                      % 用绿色虚线绘制 t-z3
>> xlabel('t(seconds)');                  % 填写 X 轴说明
>> ylabel('signals');                     % 填写 Y 轴说明
>> legend('x','z_1','z_2','z_3');         % 填写图例
>> figure;                                % 生成新图框
>> subplot(1,2,1);                        % 生成并列放置 2 个子图中的左边一个
>> plot(t,z4,'k',t,x,'b:',t,y,'r:');      % 用黑实线绘制 z4，蓝和红点线绘制 x 和 y
>> title('z_4(t)');                       % 填写标题
>> subplot(1,2,2);                        % 生成并列放置 2 个子图中的右边一个
>> plot(t,z5,'k',t,x,'b:',t,y,'r:');
>> title('z_4(t)');
```

程序运行结果如图 1.2 和 1.3 所示。

### Matlab 知识－绘图基础

上例较全面地演示了 Matlab 绘图的基本方法，其中画线函数 plot 是最基本最重要的绘图函数，它的基本调用格式是 plot(x,y,s)，其中矢量 x 和 y 分别表示线条上采样点的两个坐标值，字符串 s 是表示颜色、线型和点型的字符组合，其意义在表 1.3 中给出。

对应于用 plot 绘制连续时间信号，可以用 stem 绘制离散时间信号，请同学们结合帮助自己课下练习。

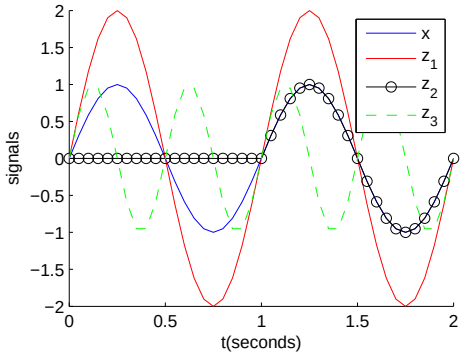


图 1.2: 例 1.4 绘第一幅图

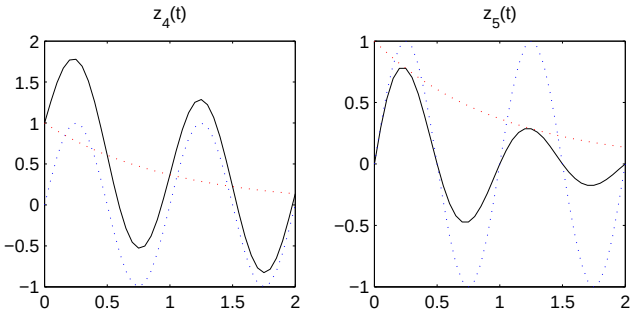


图 1.3: 例 1.4 绘第二幅图

表 1.3: plot 函数绘图类型

| 符号 | 类型 | 意义 |
|---|---|---|
| b/g/r/c/m/y/k | 颜色 | 蓝/绿/红/蓝绿/紫红/黄/黑 |
| ./x/+/h/*/s/d/v/^/</>/p/o | 点型 | 点/x/+/六角星/星号/方形/菱形/下三角/上三角/左三角/右三角/正五边形/圆圈 |
| -/:/-./– | 线型 | 实线/点线/点划线/虚线 |

Matlab 的图形窗口提供了强大的输出功能，支持多种图形格式的存储和转换。绘图结束后，如果同学们需要把图片插入到 Word 等文本编辑器中，可在绘图窗口中选择“Edit→Copy Figure”，然后在 Word 中“粘贴”即可（请复制之前勾中“Edit→Copy Options”中的“Transparent background”，这样粘贴后视觉效果更好）；如果需要把图片保存成文件，可点击“File→Save As”再保存为需要的格式，Matlab 提供了包括“TIFF”，“JPG”和“EPS”在内的十多种图像格式。

---

为节约篇幅，在本书后面的例题中，我们不再列出绘图部分的源程序，而代之以一个函数名。同学们可以从光盘中找到该函数并学习相关命令。

对于符号函数，Matlab 提供了一个功能强大的函数 ezplot 实现绘图功能。如下例。

**例 1.5** 用符号表达式法绘制函数 $x(t) = 1 + 2sin(t)$ 在 $[0, 2\pi]$ 的波形。

**解**

```
>> syms t;                    % 定义符号t
>> x = sym('1+2*sin(t)');     % 定义函数x(t) = 1 + 2 sin(t)
>> ezplot(x,[0,2*pi]);        % 绘制t从0到2π区间的x(t)
```

ezplot 也可以用于自定义的函数，只要定义好函数的输入输出关系即可。

## 第五节　阶跃信号与冲激信号

阶跃函数和冲激函数是两个特殊的数学函数，前者在零点无定义，后者在零点取值无穷大。但强大的 Matlab 分别定义了 heaviside 和 dirac 实现这两个函数。下面我们分别列出这两个函数文件的全部内容并就函数的定义和特殊数值等相关知识进行说明。

在命令窗口中输入 open heaviside 将在编辑/调试窗口中打开 heaviside 文件。如下所示。

```
function Y = heaviside(X)
%HEAVISIDE Step function.
% HEAVISIDE(X) is 0 for X < 0, 1 for X > 0, and NaN for X == 0.
% HEAVISIDE(X) is not a function in the strict sense.
% See also DIRAC.

% Copyright 1993-2003 The MathWorks, Inc.
% $Revision:  1.1.6.2 $ $Date:  2004/04/16 22:23:24 $

Y = zeros(size(X));
Y(X > 0) = 1;
Y(X == 0) = NaN;
```

文件第一行的首字 function 声明了这个文件定义的是一个函数，函数名为 heaviside，输入参数 X，返回值 Y。文件从第二行往下连续的若干带注释的行是这个函数的说明，也就是同学们在命令窗口中输入 help heaviside 看到的内容（注意空行后的将不会被看到）。最后三行是有实质意义的命令，分别解释如下

```
Y = zeros(size(X));        % 定义和 X 结构相同的全零变量
Y(X > 0) = 1;              % 定义对应于 X 大于 0 的 Y 为 1
Y(X == 0) = NaN;           %  X 等于零时 Y 非数值
```

同理我们可以 open dirac 函数来看，其中有实质意义的两行是

```
Y = zeros(size(X));        % 定义和 X 结构相同的全零变量
Y(X == 0) = Inf;           %  X 等于零时 Y 无穷大
```

NaN 和 Inf 是 Matlab 引入的两个重要常量，它使得很多没有意义的数学运算得以顺利进行，从而保证了程序不被频繁中断。表 1.4 还列出了一些重要的常量和特殊变量（包括我们前面已经用到的常量 pi）。

## 第六节　其它常用的 Matlab 命令

本章已经用到了一些简单的 Matlab 命令或函数，它们都在三个最基本的帮助主题中，分别是“general”、“elmat”和“elfun”，请同学们课下务必调用帮助浏览这三个主题中的所有命令和函数。表 1.5 中列出了若干最常用的命令。

表 1.4: 重要的 Matlab 常量和特殊变量

| 名称 | 说明 | 名称 | 说明 |
|---|---|---|---|
| ans | 默认保存最新结果的变量名 | pi | 圆周率$\pi$ |
| eps | 浮点数的相对误差 | inf | 无穷大 |
| NaN, nan | 非数 | i, j | 虚数单位 |
| realmin, realmax | 最小/最大浮点数 | bitmax | 最大正整数 |

表 1.5: Matlab 常用命令

| 命令 | 功能 | 命令 | 功能 |
|---|---|---|---|
| cd | 显示或改变工作目录 | clc | 清除命令窗口 |
| clear | 清除内存变量 | clf | 清楚图形窗口 |
| copyfile | 复制文件 | delete | 删除文件或图形对象 |
| demo | 运行实例程序 | dir, ls | 显示当前目录下文件 |
| disp | 显示变量内容 | echo | 命令窗口信息显示开关 |
| movefile | 移动文件 | open | 打开文件供编辑 |
| pack | 整理内存碎片 | pwd | 显示当前工作路径 |
| type | 显示文件内容 | who | 显示当前内存中变量 |

Matlab 提供了 save 函数将工作区中的数据保存到文件中，load 函数从文件中载入数据到工作区，其详细使用说明在表 1.6 中列出。

表 1.6: save 和 load 的使用方法

| 调用方法 | 说明 |
|---|---|
| save | 将工作区中所有变量保存到当前目录下的 matlab.mat 文件中。 |
| save var1 var2 | 将工作区中的变量 var1 和 var2 保存到 matlab.mat 文件中。 |
| save filename var1 var2 | 将工作区中的变量 var1 和 var2 保存到当前目录下的 filename.mat 文件中。 |
| load | 载入当前目录下的 matlab.mat 文件中的所有变量到工作区。 |
| load var1 var2 | 将 matlab.mat 中的变量 var1 和 var2 到工作区。 |
| load filename var1 var2 | 从 filename.mat 载入变量 var1 和 var2 到工作区。 |

上述命令都可以用函数形式实现，如 save('filename','var1','var2')，从而使编程实现灵活的文件变得非常方便。

第一篇

# 连续时间信号与系统

# 第二章　连续时间系统的时域分析

## 第一节　引言

连续时间系统的研究方法包括输入－输出法（端口描述法）和系统状态变量分析法。

首先介绍输入－输出法。“信号与系统”研究的大部分是线性时不变（ Linear time-invariant, LTI ）系统，可以用一元高阶微分方程描述。设激励信号 $e(t)$，系统响应 $r(t)$，则系统表示为

$$\begin{aligned} C_0\frac{\mathrm{d}^n}{\mathrm{d}t^n}r(t)+C_1\frac{\mathrm{d}^{n-1}}{\mathrm{d}t^{n-1}}r(t)+\cdots+C_{n-1}\frac{\mathrm{d}}{\mathrm{d}t}r(t)+C_nr(t) \\ =E_0\frac{\mathrm{d}^m}{\mathrm{d}t^m}e(t)+E_1\frac{\mathrm{d}^{m-1}}{\mathrm{d}t^{m-1}}e(t)+\cdots+E_{m-1}\frac{\mathrm{d}}{\mathrm{d}t}e(t)+E_me(t) \end{aligned}$$

在 Matlab 中，上述系统可用 tf 函数（传递函数 Transfer function 的简写）创建，使用方法为 sys = tf (b, a)，其中 $a=[C_0,C_1,\cdots,C_{n-1},C_n]$，$b=[E_0,E_1,\cdots,E_{n-1},E_n]$，分别是微分方程左侧和右侧的系数构成的行矢量，返回值 sys 即为上述 LTI 系统。

**例 2.1 (原书例 2-3 修改)** 描述如下系统

$$\frac{d^3}{dt^3}r(t)+7\frac{d^2}{dt^2}r(t)+16\frac{d}{dt}r(t)+12r(t)=e(t)$$

**解**

```
>> a = [1 7 16 12];          % 定义微分方程左侧系数的行矢量 a
>> b = [1];                  % 定义微分方程右侧系数的行矢量 b
>> sys = tf(b, a);           % 生成系统描述 sys
>> sys                       % 打印显示 sys
```

运行结果为

```
Transfer function:
          1
-----------------------
s^3 + 7 s^2 + 16 s + 12
```

可见 Matlab 以传递函数的形式描述 LTI 系统。传递函数的概念将在原书第四章第六节讲授，现在同学们可以利用电路理论学到的 p 算子理解传递函数中的符号 s 。

系统状态变量分析法将在原书第十二章详细讲解，我们这里先作简单介绍，因为在第三节计算零输入和零状态响应时必须用到状态变量的概念。前述系统是一个一元高阶微分方程，必然可以化成两个多元一阶微分方程组，其中一个表示系统状态在激励信号作用下的变化，称作状态方程：

$$\frac{\mathrm{d}}{\mathrm{d}t}\lambda(t)=\mathbf{A}\lambda(t)+\mathbf{B}\mathbf{e}(t)$$

另一个表示输出信号和系统状态及激励信号的关系，称作输出方程：

$$\mathbf{r}(t) = \mathbf{C}\lambda(t) + \mathbf{D}\mathbf{e}(t)$$

其中 $\lambda(t) = [\lambda_1, \lambda_2, \cdots, \lambda_k]^{\mathrm{T}}$ ， $\mathbf{e}(t) = [e_1, e_2, \cdots, e_m]^{\mathrm{T}}$ 和 $\mathbf{r}(t) = [r_1, r_2, \cdots, r_r]^{\mathrm{T}}$分别表示状态矢量、激励矢量和输出矢量， $\mathbf{A}_{k\times k}$ ， $\mathbf{B}_{k\times m}$ ， $\mathbf{C}_{r\times k}$ 和 $\mathbf{D}_{r\times m}$ 分别表示四个系数矩阵。

对于用状态方程和输出方程描述的系统，在 Matlab 中用 ss 函数（即状态空间 state space 的简写）创建。使用方法为 sys = ss(A, B, C, D)，其中输入四个参数即为上述四个矩阵，返回值 sys 表示该系统。

**例 2.2 (原书例 12-1 修改)** 描述如下系统

状态方程

$$\begin{bmatrix} \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \end{bmatrix} = \begin{bmatrix} -2 & 0 & -1 \\ 0 & -3 & 3 \\ 2 & -2 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & -3 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix}$$

输出方程

$$r(t) = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} e_1(t) \\ e_2(t) \end{bmatrix}$$

**解**

```
>> A = [-2,0,-1;0,-3,3;2,-2,0];        % 定义 A, B, C, D 四个矩阵
>> B = [1,0;0,-3;0,0];
>> C = [0,1,0];
>> D = [0,1];
>> sys = ss(A,B,C,D);                  % 创建系统描述 sys
>> sys                                 % 打印显示 sys
```

运行结果为

```
a =    x1  x2  x3
   x1  -2   0  -1
   x2   0  -3   3
   x3   2  -2   0
b =    u1  u2
   x1   1   0
   x2   0  -3
   x3   0   0
c =    x1  x2  x3
   y1   0   1   0
d =    u1  u2
   y1   0   1
Continuous-time model.
```

本节介绍了用两种方法创建 LTI 系统模型，在后面第 **??** 章第 **??** 节中将讲解另两种方法并对 LTI 模型做详细解释。

# 第二节　微分方程式的建立与求解

微分方程的解包括齐次解和特解两部分。齐次解即系统特征方程的根，用 roots 函数计算。使用方法 a=roots(p)，其中 p 为特征方程的系数构成的行矢量，返回值 a 是由根组成的列矢量。

**例 2.3 (原书例 2-3 )** 求微分方程

$$\frac{d^3}{dt^3}r(t)+7\frac{d^2}{dt^2}r(t)+16\frac{d}{dt}r(t)+12r(t)=e(t)$$

的齐次解。

**解** 系统特征方程为

$$a^3+7a^2+16a+12=0$$

```
>> p = [1 7 16 12];          % 定义多项式 p
>> a = roots(p);             % 求解多项式 p = 0 的根 a
>> a                         % 打印显示 a
```

运行结果为

```
a =  -3.0000
     -2.0000 + 0.0000i
     -2.0000 - 0.0000i
```

注意有两个重根 $-2$（虚部 $+0.0000i$ 和 $-0.0000i$ 的差异由数值计算的误差导致），从而写出齐次解为

$$r_h(t)=(A_1t+A_2)\,\mathrm{e}^{-2t}+A_3\mathrm{e}^{-3t}$$

特解即系统函数（微分方程）在给定激励信号作用下的输出，Matlab 提供 lsim 函数（即 LTI models simulation 的简写）对 LTI 系统进行仿真。使用方法为 y = lsim(sys, u, t)，其中 sys 表示 LTI 系统，矢量 u 和 t 分别表示激励信号的采样值和采样时间，返回值 y 为对应于上述采样时间的系统相应值。

**例 2.4 (原书例 2-4 )** 给定微分方程式

$$\frac{\mathrm{d}^2r(t)}{\mathrm{d}t^2}+2\frac{\mathrm{d}r(t)}{\mathrm{d}t}+3r(t)=\frac{\mathrm{d}e(t)}{\mathrm{d}t}+e(t)$$

如果已知: ① $e(t)=t^2$；② $e(t)=\mathrm{e}^t$，分别求两种情况下此方程的特解。

**解**

```
>> a = [1 2 3];
>> b = [1 1];
>> sys = tf(b,a);                % 创建系统描述 sys
>> t = [0:0.1:10];               % 定义仿真时间为 0 到 10 秒，采样间隔为 0.1 秒
>> e1 = t.^2;                    % 定义激励信号 e1 = t^2
>> r1 = lsim(sys,e1,t);          % 用 e1 激励 sys，输出为 r1
>> e2 = exp(t);                  % 定义激励信号 e2 = e^t
```

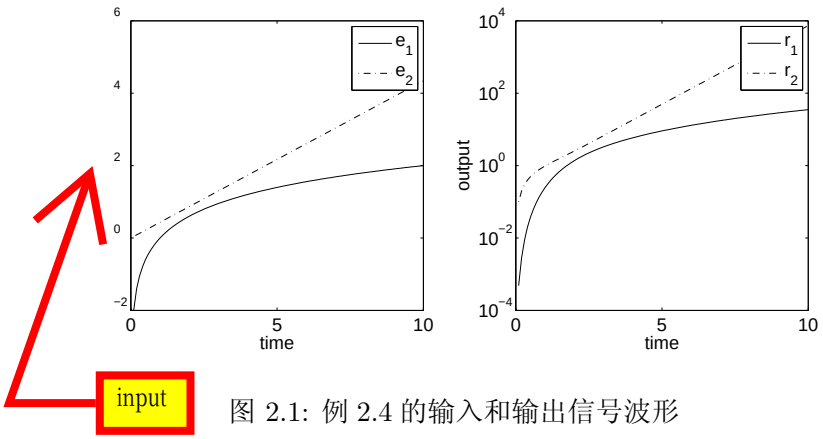


图 2.1: 例 2.4 的输入和输出信号波形

```
>> r2 = lsim(sys,e2,t);                % 用 e2 激励 sys，输出为 r2
>> ex_2_4_plot();                      % 调用函数绘制输出
```

绘制结果如图 2.1 所示。注意 Matlab 用数值方法对连续时间系统进行仿真，所以采样间隔直接影响仿真结果的准确性。同学们可以将上述脚本中采样间隔改为 1 秒，再运行程序，对比两次绘图结果有何不同。一般来说，随着采样率的提高，仿真结果和理论值会越来越相似。

---

**Matlab 知识－多项式运算**

Matlab 提供了一些处理多项式的专用函数，可以使用它们方便地求解多项式的根，对多项式进行四则运算、积分和微分等。首先，对于多项式 $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$，Matlab 约定用行矢量 $p = [a_0, a_1, \cdots, a_n]$ 来表示，这样多项式计算问题即转换为向量计算问题。

poly(A) 函数用于创建多项式：如果参数 A 是一个矢量，则返回以 A 的每个元素为根的多项式（矢量）；如果 A 是一个方阵，则返回矩阵 A 的特征多项式，即 $\det(\lambda\mathbf{I} - A)$。

若要求出多项式的值，即将 $x_0$ 代入 $p(x)$ 求解 $p(x_0)$，可以用 polyval(p,x0) 或者 polyvalm(p,x0)，前者以矩阵元素为计算单位（即将 x0 中的每个元素分别代入 $p(x)$ 的返回值再组成矩阵），后者以矩阵为计算单位（因而 x0 必须为方阵）。求多项式的根就用 roots(p) 函数，正文里已有介绍。

多项式的加法和减法可以直接对矢量进行，需要注意阶数必须相同，否则就要对低阶多项式前面做补零处理。乘法用 conv 实现，这个函数就是本章第五节将要介绍的卷积运算。除法用 deconv 实现，对应于解卷积，在原书第七章第七节讲授。求导和积分分别用 polyder 和 polyint 实现。这两个函数略显复杂，请同学们课下参考帮助练习。

---

## 第三节　零输入响应与零状态响应

在 Matlab 中，lsim 函数还可以对带有非零起始状态的 LTI 系统进行仿真，使用方法为 y = lsim(sys, u, t, x0)，其中 sys 表示 LTI 系统，矢量 u 和 t 分别表示激励信号的采样值和采样时间，矢量 x0 表示该系统的初始状态，返回值 y 是系统响应值。如果只有起始状态而没有激励信号，或者令激励信号为零，则得到零输入响应。如果既有初始状态，也有激励信号，则得到完全响应。

**请注意 lsim 函数只能对用状态方程描述的 LTI 系统仿真非零起始状态响应（对传递函数描述的**

**LTI 系统将失效**）。 Matlab 的这种约束表明了系统状态变量分析法的重要性。这也是本章开始即介绍状态方程的原因。

**例 2.5 (原书例 2-8 )** 给定图 *2.2* 所示电路，$t<0$ 开关 $S$ 处于 *1* 的位置而且已经达到稳态，将其看做起始状态，当 $t=0$ 时，$S$ 由 *1* 转向 *2*。分别求 $t>0$ 时 $i(t)$ 的零输入响应和零状态响应。

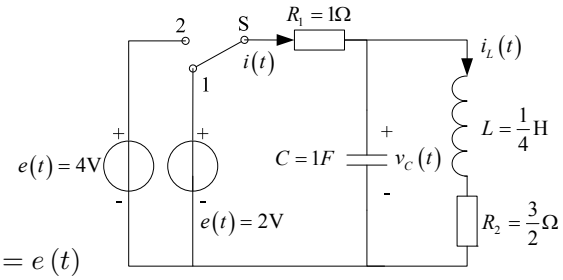


**图 2.2:** 例 2.5 电路

**解** 由所示电路写出回路方程和结点方程：

$$R_1 i(t)+v_C(t)=e(t)$$

$$v_C(t)=L\frac{\mathrm{d}}{\mathrm{d}t}i_L(t)+i_L(t)R_2$$

$$i(t)=C\frac{\mathrm{d}}{\mathrm{d}t}v_C(t)+i_L(t)$$

选择 $v_C(t)$ 和 $i_L(t)$ 作为状态变量，$i(t)$ 作为输出变量，得到

状态方程

$$\begin{bmatrix}\dot{v}_C(t)\\ \dot{i}_L(t)\end{bmatrix}=\begin{bmatrix}-\frac{1}{R_1C} & -\frac{1}{C}\\ \frac{1}{L} & -\frac{R_2}{L}\end{bmatrix}\begin{bmatrix}v_C(t)\\ i_L(t)\end{bmatrix}+\begin{bmatrix}\frac{1}{R_1C}\\ 0\end{bmatrix}e(t)$$

输出方程

$$i(t)=\begin{bmatrix}-\frac{1}{R_1} & 0\end{bmatrix}\begin{bmatrix}v_C(t)\\ i_L(t)\end{bmatrix}+\frac{1}{R_1}e(t)$$

下面我们将用两种方法计算完全响应。第一种方法：首先仿真 2V 电压作用足够长时间（ 10 秒钟）后系统进入稳态，从而得到稳态值，再以该值作为初始值仿真 4V 电压作用下的输出，即是系统的完全响应。为充分掌握 lsim 函数的使用方法，我们还仿真了系统的零状态响应和零输入响应。第二种方法：构造一个激励信号，先保持 2V 足够长时间再跳变为 4V，然后即可以零初始状态一次仿真得到系统的完全响应。

```
>> C = 1; L = 1/4; R1 = 1; R2 = 3/2;        % 定义器件参数
>> A = [-1/R1/C,-1/C;1/L,-R2/L];            % 定义 A 、B 、C 和 D 四个矩阵
>> B = [1/R1/C;0];
>> C = [-1/R1,0];
>> D = [1/R1];
>> sys = ss(A,B,C,D);                       % 创建 LTI 系统 sys
>> tn = [-10:0.01:-0.01]';                  % 生成 -10 到 0 秒的采样时间，间隔为 0.01 秒
>> en = 2*(tn<0);                           % 生成激励信号的采样值，e(t) = 2
>> [rn tn xn]= lsim(sys,en,tn);             % 仿真 t < 0 时的输出信号
>> x0 = xn(length(en),:);                   %  x0 记录了初始状态，即 0_ 状态的值
>> t = [0:0.01:10]';                        % 生成从 0 到 10 秒的采样时间，间隔为 0.01 秒
>> e = 4*(t>=0);                            % 生成激励信号的采样值，e(t) = 4
>> ezi = 0*(t>=0);                          % 生成零输入信号的采样值，e(t) = 0
>> rzs = lsim(sys,e,t);                     % 仿真零状态响应
>> rzi = lsim(sys,ezi,t,x0);                % 仿真零输入响应
```

```
>> rf = lsim(sys,e,t,x0);                % 仿真完全响应
>> r1 = lsim(sys,[en;e],[tn;t]);         % 用另一种方法直接仿真完全响应
>> ex_2_8_plot();                        % 绘制输出图形
```

两种方法的输出结果如图 2.3 所示。

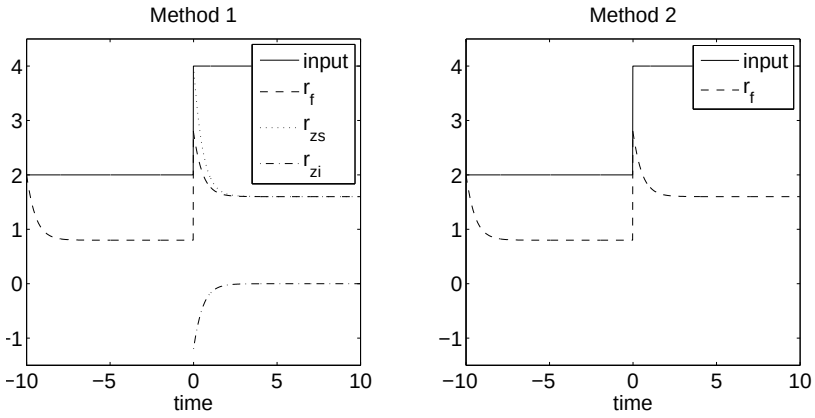


图 2.3: 例 2.5 运行结果

## 第四节　冲激响应与阶跃响应

如果分别用冲激信号和阶跃信号作激励，lsim 函数可仿真出冲激响应和阶跃响应。但鉴于这两种响应的重要性，Matlab 专门提供了 impulse(sys) 和 step(sys) 两个函数分别产生 LTI 系统的冲激响应和阶跃响应，其中 sys 表示 LTI 系统。

**例 2.6 (原书例 2-9 和 2-10 )** 对图 *2.2* 所示电路，分别求电流 $i(t)$ 对激励 $e(t)=\delta(t)$ 和 $e(t)=u(t)$ 的冲激响应 $h(t)$ 和 $g(t)$ 。

**解** 先求得电路的微分方程表示

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}i(t)+7\frac{\mathrm{d}}{\mathrm{d}t}i(t)+10i(t)=\frac{\mathrm{d}^2}{\mathrm{d}t^2}e(t)+6\frac{\mathrm{d}}{\mathrm{d}t}e(t)+4e(t)$$

在 Matlab 命令窗口中输入，

```
>> a = [1 7 10];
>> b = [1 6 4];
>> sys = tf(b,a);                % 定义LTI系统 sys
>> figure;
>> subplot(2,1,1);
>> impulse(sys);                 % 用 impulse 函数绘制系统 sys 的冲激响应
>> subplot(2,1,2);
>> step(sys);                    % 用 step 函数绘制系统 sys 的阶跃响应
```

运行结果如图 2.4 所示。在调用 impulse 或者 step 函数时如果加上返回值，例如 h = impulse(sys, t)，则 h 将被赋予冲激响应在采样时刻 t 的值，同时不再自动绘出波形。

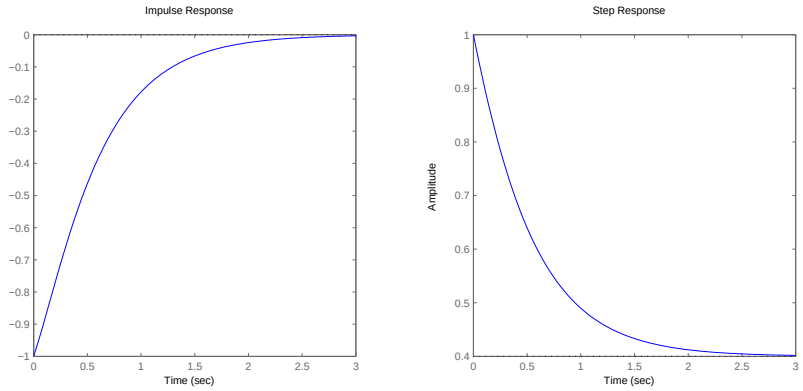


图 2.4: 例 2.6 运行结果

---

**Matlab 知识－是否需要返回值**

经过一段时间的学习，同学们可能已经深刻感受到几乎每个 Matlab 函数都存在多种调用格式，对应于不同的输入输出变量。很多函数都有这样的特点，如果你不要返回值，Matlab 会主动利用返回值帮你做一些操作，比如把返回值以图形方式表现出来，就象 impulse 一样。这样的双重功能为用户提供了很大的方便：如果只要了解返回值的特征，那就让 Matlab 画出来好了；如果关心某个具体的值，就让它算出来。似乎唯一的不足是：Matlab 自动绘制的图形比较难操纵，比如修改标题字号就有些麻烦。

---

## 第五节　卷积

在 Matlab 中，连续时间的卷积运算可以通过数值方式近似。首先对卷积公式

$$f(t) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t-\tau) \mathrm{d}\tau$$

两侧以 $T$ 为间隔采样，再将积分拆成长度为 $T$ 的若干小段，得到

$$f(nT) = \sum_{m=-\infty}^{\infty} \int_{mT}^{mT+T} f_1(\tau) f_2(nT-\tau) d\tau$$

假设采样间隔足够短，以至于 $f_1(t)$ 和 $f_2(t)$ 相邻采样点上的值几乎不变，从而近似有

$$f(nT) \approx T \sum_{m=-\infty}^{\infty} f_1(mT) f_2(nT-mT)$$

Matlab 中定义了 w=conv(u,v) 函数实现卷积和（将在原书第七章中讲授）

$$w(n) = \sum_{m} u(m) v(n+1-m).$$

对比以上两式，可发现用 conv 函数很容易实现连续时间卷积。

**例 2.7 (原书图 2-12 修改)** 用数值方法实现原书图 *2-12* 的卷积。

```
>> T = 0.01;                          % 定义采样间隔 T
>> t = -1:T:4;                        % 生成采样时间点 t
>> e = (t>-1/2&t<1);                  % 定义激励信号 e(t)，注意从 -1 开始采样
>> h = (t>0&t<2).*t/2;                % 定义冲激响应 h(t)，开始采样时间同上
>> r = T*conv(e,h);                   % 计算卷积输出 r
>> fg_2_12_plot();                    % 绘图输出
```

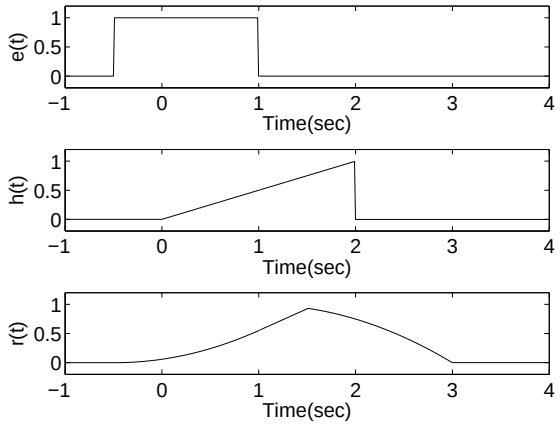


图 2.5: 例 2.7 运行结果

输出结果如图 2.5 所示。需要注意的是 conv 函数认为输入值均是从零时刻开始，所以为了绘图时对齐，需要重新调整输出的相对位置，调整方法请同学们课下自己推导。

# 第三章　傅里叶变换

我们先介绍傅里叶变换，再讲傅里叶级数。这样更有助于理解并节约篇幅。

## 第一节　傅里叶变换

Matlab 提供了强大的符号函数 fourier 和 ifourier 实现傅里叶变换和逆变换。

**例 3.1** 计算 $tu(t)$ 和 $\sin(t)$ 的傅里叶变换。

**解**

```
>> syms t                          % 定义符号 t
>> F1 = fourier(t*heaviside(t)) % 计算 tu(t) 的傅里叶变换 F1
>> F2 = fourier(sin(t));           % 计算 sin(t) 的傅里叶变换 F2
```

运行结果为

```
F1 =                              % F1 = i*(pi*dirac(1,w)*w^2+i)/w^2 = i*pi*dirac(1,w) - 1/w^2
i*(pi*dirac(1,w)*w^2+i)/w^2
F2 =                              % F2 = i*pi*(dirac(w+1)-dirac(w-1))
i*pi*(dirac(w+1)-dirac(w-1))
```

注意 heaviside 和 dirac 函数在 Matlab 中分别表示阶跃函数和冲激函数。

---

**Matlab 知识－符号运算**

在 Matlab 中，符号变量的四则运算和数值变量完全相同，注意“次幂”用“^” 符号表示。

符号变量可以组合成符号多项式，多项式的的合并同类项用 collect ，因式分解用 horner ，用 simplify 和 simple 进行化简，若要进行变量替换或者构造复合多项式，可以用 subs 和 compose 函数，而 finverse 用于求反函数。

微积分方面，用 limit 函数计算极限，diff 函数做微分和求导，jacobian 函数实现多元函数的求导，即返回雅可比矩阵；最后，用 int 进行积分运算。

Matlab 用 linsolve 求解线性方程组，fsolve 求解非线性方程组（注意必须先将该方程组定义成函数），solve 函数则用于求解一般的方程组。

微分方程组用 dsolve 实现。

---

下面我们将重点介绍用数值方法计算傅里叶变换的步骤。傅里叶变换的表达式为

$$F(\omega)=\int_{-\infty}^{\infty}f(t)\mathrm{e}^{-\mathrm{j}\omega t}\mathrm{d}t=\int_{t_1}^{t_2}f(t)\mathrm{e}^{-\mathrm{j}\omega t}\mathrm{d}t \tag{3.1}$$

其中 $[t_1,t_2]$ 为 $f(t)$ 的主要取值区间，定义 $T=t_2-t_1$ 为区间长度。在该区间内采样 $N$ 个点，采样间隔为 $\Delta t=\frac{T}{N}$，得到

$$F(\omega)=\frac{T}{N}\sum_{n=0}^{N-1}f\left(t_1+n\Delta t\right)\mathrm{e}^{-\mathrm{j}\omega(t_1+n\Delta t)}$$

用上式可以算出任意频点的傅里叶变换值。假设 $F(\omega)$ 的主要取值区间位于 $[\omega_1,\omega_2]$，我们要计算其间均匀采样的 $K$ 个值，则有

$$F(\omega_1+k\Delta\omega)=\frac{T}{N}\sum_{n=0}^{N-1}f\left(t_1+n\Delta t\right)\mathrm{e}^{-\mathrm{j}(\omega_1+k\Delta\omega)(t_1+n\Delta t)} \tag{3.2}$$

其中$\Delta\omega=\frac{\Omega}{K}$为频域采样间隔，$\Omega=\omega_2-\omega_1$ 为带宽。

下面利用同样的方法研究傅里叶逆变换，由于 $f(t)$ 的频带主要位于 $[\omega_1,\omega_2]$，近似有

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty}F(\omega)\mathrm{e}^{\mathrm{j}\omega t}\mathrm{d}\omega=\frac{\Omega}{2\pi K}\sum_{k=0}^{K-1}F(\omega_1+k\Delta\omega)\mathrm{e}^{\mathrm{j}(\omega_1+k\Delta\omega)t} \tag{3.3}$$

我们仍然只关心时域采样点上的值，得到

$$f(t_1+n\Delta t)=\frac{\Omega}{2\pi K}\sum_{k=0}^{K-1}F(\omega_1+k\Delta\omega)\mathrm{e}^{\mathrm{j}(\omega_1+k\Delta\omega)(t_1+n\Delta t)} \tag{3.4}$$

现在即可用式 (3.2) 计算傅里叶变换的 $K$ 个频域采样值，进而可以用式 (3.4) 恢复时域信号。通过下例具体讲解利用该公式的三种方法，同时介绍用矩阵代替循环加速 Matlab 运算的技巧。

**例 3.2 (原书图 3-21 修改)** 请绘制矩形脉冲

$$f(t)=\begin{cases}1 & |\,t\,|<\frac{1}{2}\\ 0 & \text{otherwise}\end{cases}$$

的波形 ($t\in[-1,1]$) 和频谱 $F(\omega)$ ($\omega\in[-8\pi,8\pi]$)，并利用你计算得到的频谱恢复时域信号 $f_s(t)$，比较和原信号 $f(t)$ 的差别。

**解 方法一：直接计算法**

直接编程实现式 (3.2) 和 (3.4)，对傅里叶变换和逆变换分别用二重循环实现。

```
>> T = 2;                                  % 定义时域采样区间长度
>> N = 200;                                % 定义时域采样点数
>> t = linspace(-T/2,T/2-T/N,N)';          % 定义时域采样点
>> f = 0*t;                                % 初始化时域信号
>> f(t>-1/2&t<1/2) = 1;                    % 为时域信号赋值
```

```
>> OMG = 16*pi;                                  % 定义频域采样区间长度
>> K = 100;                                      % 定义频域采样点数
>> omg = linspace(-OMG/2,OMG/2-OMG/K,K)';        % 定义频域采样点
>> F = 0*omg;                                    % 初始化频谱
>> for k = 1:K                                   % 循环计算每个频域采样点的频谱
>>   for n = 1:N                                 % 用循环实现式 (3.2) 中的求和运算
>>     F(k) = F(k) + T/N*f(n)*exp(-j*omg(k)*t(n));
>>   end
>> end
>> fs = 0*t;                                     % 初始化合成信号
>> for n = 1:N                                   % 循环计算每个时域采样点的合成信号
>>   for k = 1:K                                 % 用循环方式实现式 (3.4) 中的求和运算
>>     fs(n) = fs(n) + OMG/2/pi/K*F(k)*exp(j*omg(k)*t(n));
>>   end
>> end
>> fg_3_21_direct_plot();                        % 绘制输出图形
```

结果如图 3.1 所示。由于恢复时域信号时丢弃了高频分量，因而合成信号和原始信号相比，在跳变处有较大的波动，这就是 Gibbs 现象。

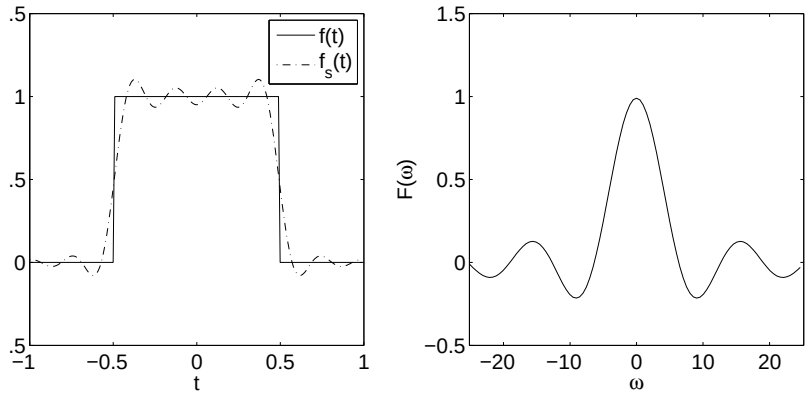


图 3.1: 例 3.2 的波形和频谱

上述方法的优点是道理直观，编程简便，缺点是计算量大，耗时较多。我们知道 Matlab 对矢量和矩阵运算做了很大的优化，所以如果可以将式 (3.2) 和 (3.4) 中的循环改为矢量运算必将极大的降低计算复杂度。

## Matlab 知识－控制命令

和所有程序语言一样，在顺序执行的基础上，Matlab 还提供分支和循环控制语句。

分支语句包括 if - elseif - else - end 和 switch - case - otherwise -end，其中 if 和 elseif 后可跟随逻辑变量或数值变量（非“0”表示真“0”表示假），也可以是符号变量（要求能够转化为数值变量）；switch 后跟随变量名，case 后跟随常量，这些变量或者常量同样可以是数值、逻辑或者符号变（常）量。

循环语句包括 for -end和 while -end，其中可以用 continue 提前进入下一次迭代或者用 break 跳出循环体。while 后内容和 if 相同，不必解释。最复杂的是 for，一般使用方法是

```
>> for 循环变量名 = 起始值 : 增量 : 结束值
```

事实上，Matlab 只要求 = 后面跟一个矢量，而各元素间不必等间隔，例如

```
>> for n=[1,2,3,8]
```

即可实现次数为 4 的循环，其中 n 依次取值为 1，2，3 和 8。

最后，Matlab 还支持 try - catch -end 语句捕获错误事件。这个并不常用，请同学们参考帮助自学。

---

**方法二：矢量计算法**

式 (3.2) 中的累加和可以用内积实现，即一个行矢量左乘一个列矢量相乘，我们将已知时域采样值 $f(t_1), f(t_1+\Delta t), \cdots, f(t_2-\Delta t)$ 写成列矢量，则式 (3.2) 改写为

$$F(\omega_1+k\Delta\omega)=\frac{T}{N}\left[\begin{array}{cccc} \mathrm{e}^{-\mathrm{j}(\omega_1+k\Delta\omega)t_1} & \mathrm{e}^{-\mathrm{j}(\omega_1+k\Delta\omega)(t_1+\Delta t)} & \cdots & \mathrm{e}^{-\mathrm{j}(\omega_1+k\Delta\omega)(t_2-\Delta t)} \end{array}\right]\left[\begin{array}{c} f(t_1) \\ f(t_1+\Delta t) \\ \vdots \\ f\left(t_2-\Delta t\right) \end{array}\right] \tag{3.5}$$

同样，式 (3.4) 改写为

$$f(t_1+n\Delta t)=\frac{\Omega}{2\pi K}\left[\begin{array}{cccc} \mathrm{e}^{\mathrm{j}\omega_1(t_1+n\Delta t)} & \mathrm{e}^{\mathrm{j}(\omega_1+\Delta\omega)(t_1+n\Delta t)} & \cdots & \mathrm{e}^{\mathrm{j}(\omega_2-\Delta\omega)(t_1+n\Delta t)} \end{array}\right]\left[\begin{array}{c} F(\omega_1) \\ F(\omega_1+\Delta\omega) \\ \vdots \\ F\left(\omega_2-\Delta\omega\right) \end{array}\right] \tag{3.6}$$

```
>> F = 0*omg;                                  % 初始化频谱
>> for k = 1:K                                 % 循环计算每个频域采样点的频谱
>>   F(k) = T/N*exp(-j*omg(k)*t).'*f;
>> end
>> fs = 0*t;                                   % 初始化合成信号
>> for n = 1:N                                 % 循环计算每个时域采样点的合成信号
>>   fs(n) = OMG/2/pi/K*exp(j*omg*t(n)).'*F;
>> end
```

上述方法用矢量相乘代替了内部循环运算，但是仍然有一个外部循环。下面我们进一步用矩阵运算把外部循环也去掉。

**方法三：矩阵计算法**

由式 (3.5) 和 (3.6) 可见，我们事实上要在时域和频域上的两个矢量之间互求，因而可用矩阵形式表述为

$$\left[\begin{array}{c} F(\omega_1) \\ F(\omega_1+\Delta\omega) \\ \vdots \\ F\left(\omega_2-\Delta\omega\right) \end{array}\right]=\frac{T}{N}\left[\begin{array}{cccc} \mathrm{e}^{-\mathrm{j}\omega_1 t_1} & \mathrm{e}^{-\mathrm{j}\omega_1(t_1+\Delta t)} & \cdots & \mathrm{e}^{-\mathrm{j}\omega_1(t_2-\Delta t)} \\ \mathrm{e}^{-\mathrm{j}(\omega_1+\Delta\omega)t_1} & \mathrm{e}^{-\mathrm{j}(\omega_1+\Delta\omega)(t_1+\Delta t)} & \cdots & \mathrm{e}^{-\mathrm{j}(\omega_1+\Delta\omega)(t_2-\Delta t)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{e}^{-\mathrm{j}(\omega_2-\Delta\omega)t_1} & \mathrm{e}^{-\mathrm{j}(\omega_2-\Delta\omega)(t_1+\Delta t)} & \cdots & \mathrm{e}^{-\mathrm{j}(\omega_2-\Delta\omega)(t_2-\Delta t)} \end{array}\right]\left[\begin{array}{c} f(t_1) \\ f(t_1+\Delta t) \\ \vdots \\ f\left(t_2-\Delta t\right) \end{array}\right] \tag{3.7}$$

$$\begin{bmatrix} f(t_1) \\ f(t_1+\Delta t) \\ \vdots \\ f\left(t_2-\Delta t\right) \end{bmatrix} = \frac{\Omega}{2\pi K} \begin{bmatrix} \mathrm{e}^{\mathrm{j}\omega_1 t_1} & \mathrm{e}^{\mathrm{j}(\omega_1+\Delta\omega)t_1} & \cdots & \mathrm{e}^{\mathrm{j}(\omega_2-\Delta\omega)t_1} \\ \mathrm{e}^{\mathrm{j}\omega_1(t_1+\Delta t)} & \mathrm{e}^{\mathrm{j}(\omega_1+\Delta\omega)(t_1+\Delta t)} & \cdots & \mathrm{e}^{\mathrm{j}(\omega_2-\Delta\omega)(t_1+\Delta t)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{e}^{\mathrm{j}\omega_1(t_2-\Delta t)} & \mathrm{e}^{\mathrm{j}(\omega_1+\Delta\omega)(t_2-\Delta t)} & \cdots & \mathrm{e}^{\mathrm{j}(\omega_2-\Delta\omega)(t_2-\Delta t)} \end{bmatrix} \begin{bmatrix} F(\omega_1) \\ F(\omega_1+\Delta\omega) \\ \vdots \\ F\left(\omega_2-\Delta\omega\right) \end{bmatrix} \tag{3.8}$$

简写为

$$\mathbf{F} = \frac{T}{N}\mathbf{U}\mathbf{f} \tag{3.9}$$

$$\mathbf{f} = \frac{\Omega}{2\pi K}\mathbf{V}\mathbf{F} \tag{3.10}$$

其中 $\mathbf{U}$ 和 $\mathbf{V}$ 分别代表式 (3.7) 和 (3.8) 中的变换矩阵。

现在一个新的问题出现了：如何有效的构造矩阵 $\mathbf{W}$ ？ Matlab 提供了 kron 函数（即张量积）巧妙的解决了这个问题，请同学们查看帮着理解该函数的使用方法。

```
>> U = exp(-j*kron(omg,t.'));          % 定义变换矩阵
>> F = T/N*U*f;                        % 左乘矩阵实现傅里叶变换
>> V = exp(j*kron(t,omg.'));           % 定义逆变换矩阵
>> fs = OMG/2/pi/K*V*F;                % 左乘逆变换矩阵实现傅里叶逆变换
```

前面我们给出了实现傅里叶变换和逆变换的三种方法，其中最后一种矩阵计算方法在讲离散系统时还会有介绍，届时我们可以用更简单的 fft 函数实现。

同学们在本科高年级或者研究生课程中选修《泛函分析》等课程，会讲到线性算子等于矩阵算子，即线性变换都可以用矩阵形式表征。同学们在以后编程时一定要牢记这一点，首先把待实现算法写成矩阵形式，然后再用 Matlab 直接实现，切忌使用循环等外部迭代方式!

### Matlab 知识－程序优化

Matlab 基于逐条解释命令的方法执行（尽管 7.0 版也支持编译出独立于开发环境的可执行文件，但事实上很少用到），因而对循环和判断等此类需要多条命令语句组合实现的代码的运行效率是非常低的。与此相对应的是，Matlab 矩阵运算做了大量的优化，很多矩阵相关的函数都用低级语言编写并针对处理器特性进行了优化，因而运行效率很高。所以优化 Matlab 代码的首要原则就是尽量少使用程序控制命令，尽量多用矩阵和向量操作实现。对下述循环

```
>> for n=1:1e6
>>   sin(n);
>> end
```

Matlab 需要运行大约 2 秒种才能重现提示符进入准备状态（处理器 Inter P IV 2.4G，内存 512M，下同），而

```
>> sin([1:1e6]);
```

几乎是一瞬间即可执行结束。

由于对代码进行解释执行，Matlab 不会对变量预先分配内存，因而也不要求程序中对变量进行预先定义，这种灵活的方式方便了用户编程，但同时带来潜在的低效率问题。例如下面代码（注意我们先用 clear 释放所有内存变量）

```
>> clear all
>> x=zeros(1e6,1);
```

```
>> for n=1:1e6
>>   x(n)=sin(n);
>> end
```

同样在 2 秒种后返回，而

```
>> clear all
>> for n=1:1e6
>>   x(n)=sin(n);
>> end
```

则要运行至少 1 分钟（我没有耐心等它运行完）。后一段程序效率低的原因就是没有对 x 做初始化，每次都要为 x 重新分配内存，因而效率极低。

矩阵化和变量预定义是提高 Matlab 代码效率的主要手段，同学们一定要认真掌握。但同时需要注意，Matlab 对矩阵的优化只做到了 2 维，因而巧妙地把算法设计成三维矩阵实现也是意义不大的。最后再说一句程序优化的必要性：尽管 Matlab 支持用 Ctrl+C 中断当前程序的运行，但往往不管用!

---

## 第二节　周期信号的傅里叶级数分析

周期为 $T_1$ 的连续信号 $f(t)$ 的指数形式傅里叶级数为

$$f(t)=\sum_{k=-\infty}^{\infty}F_k\mathrm{e}^{\mathrm{j}k\omega_1 t} \tag{3.11}$$

其中 $\omega_1=\dfrac{2\pi}{T_1}$，系数

$$F_k=\frac{1}{T_1}\int_{t_0}^{t_0+T_1}f(t)\mathrm{e}^{-\mathrm{j}k\omega_1 t}\mathrm{d}t \tag{3.12}$$

上述信号的三角函数形式的傅里叶级数为

$$f(t)=a_0+\sum_{k=1}^{\infty}\left[a_k\cos(k\omega_1 t)+b_k\sin(k\omega_1 t)\right] \tag{3.13}$$

其中系数

$$a_0=F_0\qquad\begin{cases}a_k=F_k+F_{-k}\\ b_k=\mathrm{j}\left(F_k-F_{-k}\right)\end{cases}\qquad(k=1,2,\cdots) \tag{3.14}$$

我们可以先对式 (3.11) 和 (3.12) 做离散化有限项表示，然后再利用上一节介绍的矩阵表示方法计算傅里叶级数的指数形式的系数和展开式，最后再进一步表示出三角函数形式级数的系数。

首先假设可以用 $F_{k_1},F_{k_1+1},\cdots,F_{k_2}$ 共 $K$ 项近似原函数，即

$$f(t)=\sum_{k=k_1}^{k_2}F_k\mathrm{e}^{\mathrm{j}k\omega_1 t} \tag{3.15}$$

将我们关心的从 $t_0$ 到 $t_0+T$ 之间的 $N$ 个采样值

$$f(t_0+n\Delta t)=\sum_{k=k_1}^{k_2}F_k\mathrm{e}^{\mathrm{j}k\omega_1(t_0+n\Delta t)} \tag{3.16}$$

写成矩阵形式为

$$\begin{bmatrix} f(t_0) \\ f(t_0+\Delta t) \\ \vdots \\ f\left(t_0+T_1-\Delta t\right) \end{bmatrix} = \begin{bmatrix} \mathrm{e}^{\mathrm{j}k_1\omega_1 t_0} & \mathrm{e}^{\mathrm{j}(k_1+1)\omega_1 t_0} & \cdots & \mathrm{e}^{\mathrm{j}k_2\omega_1 t_0} \\ \mathrm{e}^{\mathrm{j}k_1\omega_1 (t_0+\Delta t)} & \mathrm{e}^{\mathrm{j}(k_1+1)\omega_1 (t_0+\Delta t)} & \cdots & \mathrm{e}^{\mathrm{j}k_2\omega_1 (t_0+\Delta t)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{e}^{\mathrm{j}k_1\omega_1 (t_0+T_1-\Delta t)} & \mathrm{e}^{\mathrm{j}(k_1+1)\omega_1 (t_0+T_1-\Delta t)} & \cdots & \mathrm{e}^{\mathrm{j}k_2\omega_1 (t_0+T_1-\Delta t)} \end{bmatrix} \begin{bmatrix} F_{k_1} \\ F_{k_1+1} \\ \vdots \\ F_{k_2} \end{bmatrix} \tag{3.17}$$

离散化数值近似后，式 (3.12) 变为

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f(t_0+n\Delta t)\mathrm{e}^{-\mathrm{j}k\omega_1(t_0+n\Delta t)} \tag{3.18}$$

写成矩阵形式

$$\begin{bmatrix} F_{k_1} \\ F_{k_1+1} \\ \vdots \\ F_{k_2} \end{bmatrix} = \frac{1}{N}\begin{bmatrix} \mathrm{e}^{-\mathrm{j}k_1\omega_1 t_0} & \mathrm{e}^{-\mathrm{j}k_1\omega_1 (t_0+\Delta t)} & \cdots & \mathrm{e}^{-\mathrm{j}k_1\omega_1 (t_0+T_1-\Delta t)} \\ \mathrm{e}^{-\mathrm{j}(k_1+1)\omega_1 t_0} & \mathrm{e}^{-\mathrm{j}(k_1+1)\omega_1 (t_0+\Delta t)} & \cdots & \mathrm{e}^{-\mathrm{j}(k_1+1)\omega_1 (t_0+T_1-\Delta t)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{e}^{-\mathrm{j}k_2\omega_1 t_0} & \mathrm{e}^{-\mathrm{j}k_2\omega_1 (t_0+\Delta t)} & \cdots & \mathrm{e}^{-\mathrm{j}k_2\omega_1 (t_0+T_1-\Delta t)} \end{bmatrix} \begin{bmatrix} f(t_0) \\ f(t_0+\Delta t) \\ \vdots \\ f\left(t_0+T_1-\Delta t\right) \end{bmatrix} \tag{3.19}$$

对比矩阵化的傅里叶级数的数值表达式和傅里叶变换的数值表达式，可以发现其形式非常相似，因而编程实习方法也大致相同。

**例 3.3 (原书图 3-6 修改)** *绘制周期 $T_1 = 1$ 幅度 $E = 1$ 的对称方波的前 10 项傅里叶级数的系数 ( 三角函数形式 ), 并用前 5 项恢复原信号。*

**解** 如前所述，我们计算三角函数形式的傅里叶级数的系数可以先计算指数形式的系数，再转化成三角函数形式。由三角函数形式的系数合成原函数时，我们则直接合成，本题中函数为偶对称，正弦分量全部为零，由式 (3.13) 可写出前 $K$ 项级数的合成公式

$$\begin{bmatrix} f(t_0) \\ f(t_0+\Delta t) \\ \vdots \\ f(t_0+T_1-\Delta t) \end{bmatrix} = \begin{bmatrix} 1 & \cos(\omega_1 t_0) & \cdots & \cos(K\omega_1 t_0) \\ 1 & \cos(\omega_1 (t_0+\Delta t)) & \cdots & \cos(K\omega_1 (t_0+\Delta t)) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(\omega_1 (t_0+T_1-\Delta t)) & \cdots & \cos(K\omega_1 (t_0+T_1-\Delta t)) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_K \end{bmatrix} \tag{3.20}$$

```
>> E = 1;                                  % 定义方波幅度
>> T1 = 1;                                 % 定义方波周期
>> omg1 = 2*pi/T1;                         % 定义基频
>> N = 1000;                               % 定义时域采样点数
>> t = linspace(-T1/2,T1/2-T1/N,N)';       % 生成时域采样点
>> f = 0*t;                                % 初始化时域信号
>> f(:)  = -E/2;
>> f(t>-T1/4&t<T1/4) = E/2;
>> k1 = -10;                               % 确定系数的起止下标
>> k2 = 10;
>> k = [k1:k2]';                           % 生成系数下标序列
```

```
>> F = 1/N*exp(-j*kron(k*omg1,t.'))*f;          % 求指数形式傅里叶级数的系数
>> a0 = F(11);                                  % 转换到三角函数形式的系数
>> ak = F(12:21) + F(10:-1:1);
>> fs = cos(kron(t,[0:5]*omg1))*[a0;ak(1:5)];   % 用前五个系数合成原函数
>> fg_3_6_plot();                               % 绘制图形
```

绘图输出结果如图 3.2 所示，其中 $f_k(t) = a_k \cos(k\omega_1 t)$ 。

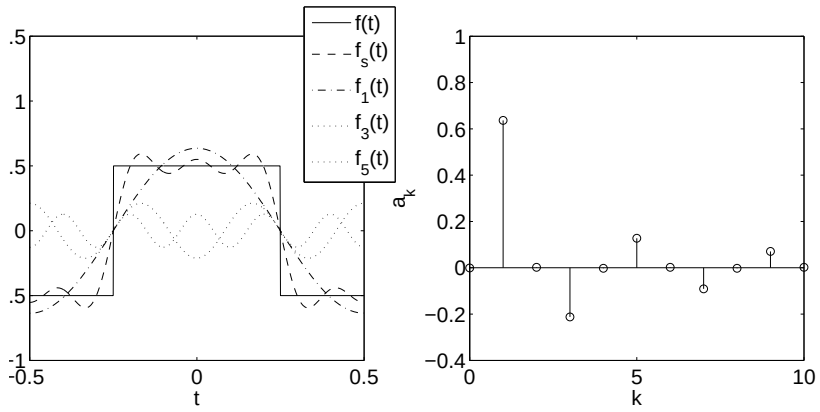


图 3.2: 例 3.3 时域波形和傅里叶级数

## 第三节　卷积特性（卷积定理）

下面用例题验证卷积定理：时域卷积对应于频域相乘。

**例 3.4 (原书例 3-9 )** 已知

$$f(t) = \begin{cases} E\left(1 - \dfrac{2\mid t \mid}{\tau}\right) & \mid t \mid \leq \dfrac{\tau}{2} \\ 0 & \mid t \mid > \dfrac{\tau}{2} \end{cases}$$

利用卷积定理求三角脉冲的频谱。

**解** 我们用两种方法计算三角脉冲的频谱：一是直接对三角脉冲做傅里叶变换；二是利用卷积定理（矩形脉冲的卷积是三角脉冲），先计算矩形脉冲的频谱，再取其平方。我们将两种计算结果绘制在一起，验证卷积定理的正确性。

```
>> N = 500;
>> K = 500;
>> OMG = 50;
>> T = 2;
>> tau = 1;
>> E = 1;
```

```
>> t = linspace(-T/2,T/2-T/K,N)';
>> g = sqrt(2*E/tau)(t>-tau/4&t<tau/4);          % 由定义生成 g(t) 的采样点
>> f = E*(1-2*abs(t)/tau).*(t>-tau/2&t<tau/2);   % 由定义生成 f(t) 的采样点
>> omg = linspace(-OMG/2,OMG/2-OMG/N,K)';
>> U = exp(-j*kron(omg,t.'));                    % 生成变换矩阵 U
>> V = exp(j*kron(t,omg.'));                     % 生成逆变换矩阵 V
>> G = T/N*U*g;                                  % 由定义计算 G(ω)
>> F = T/K*U*f;                                  % 由定义计算 F(ω)
>> Fe = G.*G;                                    % 由卷积定理计算 Fe(ω)
>> fe = OMG/2/pi/K*V*Fe;                         % 由 Fe(ω) 逆变换得到 fe(t)
>> ex_3_9_plot();                                % 绘制输出波形
```

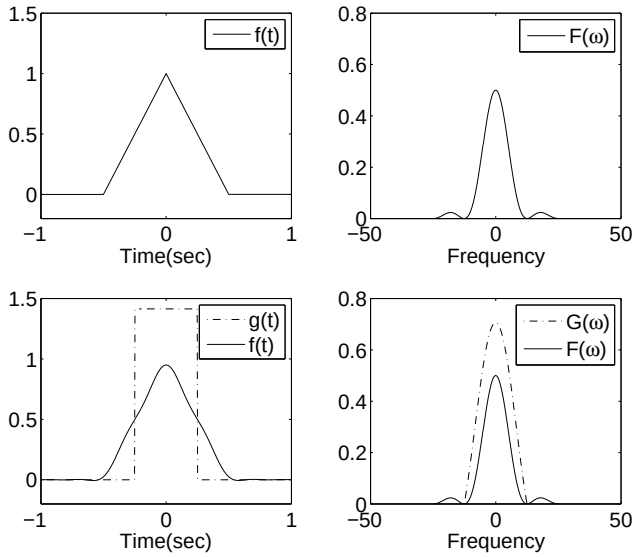


图 3.3: 例 3.4 运行结果

计算结果如图 3.3 所示。可见利用卷积定理计算得到的三角脉冲的频谱和直接计算得到的频谱是完全相同的。

# 第四章　拉普拉斯变换、连续时间系统的 s 域分析

## 第一节　拉普拉斯变换和逆变换

Matlab 用符号函数 laplace 和 ilaplace 实现拉氏变换和逆变换。

**例 4.1** 计算 $t^3$ 和 $\sin(\omega t)$ 的傅里叶变换。

**解**

```
>> syms t w                          % 定义符号 t , ω
>> F1 = laplace(t^3)                 % 计算 t^3 的傅里叶变换 F1
>> F2 = laplace(sin(w*t))            % 计算 sin(ωt) 的傅里叶变换 F2
```

运行结果为

```
F1 =                                 % F1 = 6/s^4
 6/s^4
F2 =                                 % F2 = w/(s^2+w^2)
 w/(s^2+w^2)
```

**例 4.2 (原书例 4-8 )** 求下示函数的逆变换

$$F(s)=\frac{10(s+2)(s+5)}{s(s+1)(s+3)}$$

**解**

```
>> syms s;                                          % 定义符号 s
>> f = ilaplace(10*(s+2)*(s+5)/s/(s+1)/(s+3))       % 求拉氏逆变换
```

运行结果为

```
f =                                                 % f = -10/3*exp(-3*t)-20*exp(-t)+100/3
 -10/3*exp(-3*t)-20*exp(-t)+100/3
```

用符号函数求拉氏变换和逆变换很简单，但有较大的局限性，而且无助于理解概念。我们推荐在 Matlab 协助下用部分分式展开的方法求解拉氏逆变换。Matlab 提供了部分分式展开的函数 residue ，下面分极点实数且无重根、有共轭复数极点和有多重极点三种情况分别介绍该函数使用方法。

**例 4.3 (原书例 4-9 )** 求下示函数的逆变换

$$F(s)=\frac{s^3+5s^2+9s+7}{(s+1)(s+2)}$$

**解**

```
>> b = [1,5,9,7];            % F(s) 分子多项式的系数
>> a1 = [1,1];               % F(s) 分母多项式第一个分式的系数
>> a2 = [1,2];               % F(s) 分母多项式第二个分式的系数
>> a = conv(a1,a2);          % 计算 F(s) 分母多项式的系数
>> [r,p,k] = residue(b,a)    % 部分分式展开，得到常数项 r，极点（特征根）p 和直接项 k
```

输入结果如下

```
r = -1                       % 两个部分分式常数项
 2
p = -2                       % 两个极点（特征根）
 -1
k = 1 2                      % 直接项
```

将常数与极点配对得到 $F(s)$ 的部分分式展开形式

$$F(s)=s+2-\frac{1}{s+2}+\frac{2}{s+1}$$

从而直接写出逆变换式

$$f(t)=\delta'(t)+2\delta(t)-\mathrm{e}^{-2t}+2\mathrm{e}^{-t}\qquad(t\geq 0)$$

**例 4.4 (原书例 4-10 )** 求下示函数的逆变换

$$F(s)=\frac{s^2+3}{(s^2+2s+5)(s+2)}$$

**解**

```
>> b = [1,0,3];              % F(s) 分子多项式的系数
>> a1 = [1,2,5];             % F(s) 分母多项式第一个分式的系数
>> a2 = [1,2];               % F(s) 分母多项式第二个分式的系数
>> a = conv(a1,a2);          % 计算 F(s) 分母多项式的系数
>> [r,p,k] = residue(b,a)    % 部分分式展开，得到常数项 r，极点（特征根）p 和直接项 k
```

输入结果如下

```
r =                          % 三个部分分式常数项
 -0.2000 + 0.4000i
 -0.2000 - 0.4000i
 1.4000
p =                          % 三个极点（特征根）
 -1.0000 + 2.0000i
 -1.0000 - 2.0000i
 -2.0000
```

```
k =                                    % 直接项为空，因为分子阶数小于分母阶数
  []
```

将常数与极点配对得到 $F(s)$ 的部分分式展开形式

$$F(s)=\frac{-0.2+j0.4}{s+1-j2}+\frac{-0.2-j0.4}{s+1+j2}+\frac{1.4}{s+2}$$

由原书公式 (4-57)

$$\mathcal{L}^{-1}\left[\frac{A+jB}{s+\alpha-j\beta}+\frac{A-jB}{s+\alpha+j\beta}\right]=2\mathrm{e}^{-\alpha t}\left[A\cos(\beta t)-B\sin(\beta t)\right]$$

可直接写出逆变换式

$$f(t)=1.4\mathrm{e}^{-2t}-2\mathrm{e}^{-t}\left[0.2\cos(2t)+0.4\sin(2t)\right]\qquad(t\geq 0)$$

**例 4.5 (原书例 4-12 )** 求下示函数的逆变换

$$F(s)=\frac{s-2}{s\left(s+1\right)^{3}}$$

**解**

```
>> b = [1,-2];                          % F(s) 分子多项式的系数
>> a1 = [1,0];                          % F(s) 分母多项式第一个分式的系数
>> a2 = [1,1];                          % F(s) 分母多项式第二个分式的系数
>> a = conv(conv(a1,a2),conv(a2,a2));   % 计算 F(s) 分母多项式的系数
>> [r,p,k] = residue(b,a)               % 部分分式展开
```

输入结果如下

```
r =                                    % 四个部分分式常数项
 2.0000
 2.0000
 3.0000
 -2.0000
p =                                    % 四个极点（或者说一重、三重各一个）
 -1.0000
 -1.0000
 -1.0000
 0
k =                                    % 直接项为空，因为分子阶数小于分母阶数
  []
```

注意有一个三重极点，将常数与极点配对得到 $F(s)$ 的部分分式展开形式

$$F(s)=\frac{2}{s+1}+\frac{2}{(s+1)^2}+\frac{3}{(s+1)^3}-\frac{2}{s}$$

从而直接写出逆变换式

$$f(t)=2\mathrm{e}^{-t}+2t\mathrm{e}^{-t}+\frac{3}{2}t^2\mathrm{e}^{-t}-2\qquad(t\geq 0)$$

注意 residue 输出结果中重根对应的多项式以升幂顺序排列。

用数值近似计算拉氏变换或者逆变换的方法和傅里叶变换相似，但这样做意义不大，因为我们不会对一段信号采样做拉氏变换分析其性能。

# 第二节　系统函数（网络函数）$H(s)$

回顾前面介绍到的 tf 函数，可知 Matlab 是用系统函数 $H(s)$ 描述 LTI 系统的。

**例 4.6 (原书例 4-17 )** 图 *4.1* 所示电路在 $t=0$ 时开关 $S$ 闭合，接入信号源 $e(t)=\sin(3t)$，电感起始电流等于零，求电流 $i(t)$ 。

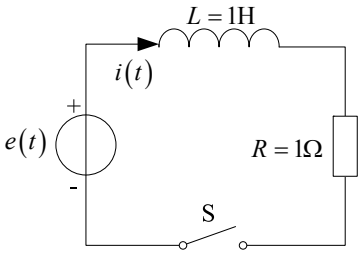


**图 4.1:** 例 4.6 电路

**解** 由电路可知传递函数为

$$H(s)=\frac{1}{s+1}$$

```
>> sys = tf(1,[1 1]);          % 定义系统传递函数 H(s)
>> t = [0:0.01:10];            % 定义采样时间 t
>> e = sin(3*t);               % 生成激励信号 e(t)
>> i = lsim(sys, e, t);        % 仿真电流信号 i(t)
>> ex_4_17_plot();             % 绘制激励和电流
```

输出结果如图 4.2 所示。

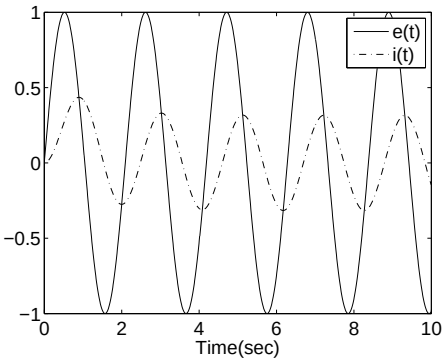


图 4.2: 例 4.6 运行结果

# 第三节　由系统函数零、极点分布决定时域特性

计算零极点可以用 roots 函数，若参数为传递函数 $H(s)$ 的分子多项式系数 $b$，则得到零点；若为分母多项式系数 $a$，则得到极点。Matlab 还提供了 zero(sys) 和 pole(sys) 函数直接计算零极点，其中 sys 表示系统传递函数。另外，[p,z]=pzmap(sys) 函数也具有计算极点 p 和零点 z 的功能；不带返回值的 pzmap(sys) 则绘制出系统的零极点图。

我们知道，零极点和传递函数的多项式系数一样，可作为 LTI 系统的描述方法。Matlab 提供了 (b,a)= zp2tf(z,p,k) 和 [z,p,k]= tf2zp b,a两个函数用于在上述两种描述方法之间进行转换，其中 k 表示用零极点表示传递函数时的系统增益。

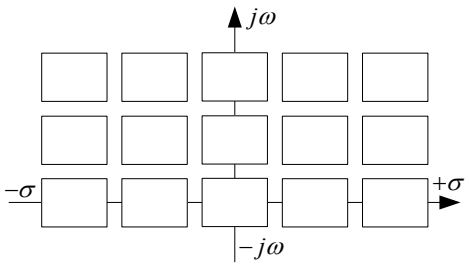


**图 4.3:** 例 4.7 框图,

**例 4.7 (原书习题 4-22 修改)** 当 $F(s)$ 极点（一阶）落于图 *4.3* 所示 $s$ 平面图中各方框所处位置时，画出对应的 $f(t)$ 波形填入方框中。

**解**

```
>> t = [0:.1:40];                 % 定义采样时间
>> figure, id = 1;                % 生成新图框，定义当前子图标号 id = 1
>> for omega = .5:-.25:0          % 循环 ω = [0.5, 0.25, 0]
>>   for sigma = -.06:.03:.06     % 循环 σ = [−0.06, −0.03, ··· , 0.06]
>>     p = sigma + j*omega;       % 定义极点 p = σ + jω
>>     if omega ~= 0              % 如果极点不在实轴上
>>       p = [p;p'];              % 则再添加一个共轭极点
>>     end
>>     [b a] = zp2tf([],p,1);     % 由零极点确定的模型转换为传递函数的多项式系数
>>     subplot(3,5,id);           % 生成标号为 id 的子图框
>>     impulse(b,a,t);            % 绘制采样时间为 t 的冲激响应
>>     set(gca,'YLim',[-20,20]);% 设置 Y 轴范围以求最好视觉效果
>>     id = id + 1;               % 子图标号加 1
>>   end                          % 内层 σ 循环结束
>> end                            % 外层 ω 循环结束
```

输出结果如图 4.4 所示。可见随着极点从虚轴左侧向右移动到虚轴右侧，其冲激响应由衰减变为发散；随着极点由实轴向上下两侧移动，冲激响应由单调变化转为振荡变化。

## 第四节　由系统函数零、极点分布决定频域特性

Matlab 提供了 freqs(b,a) 函数用于绘制系统的频率响应，包括幅度响应和相位响应，其中 $b$ 和 $a$ 分别是传递函数的分子和分母多项式系数。如果将调用方式改为 H= freqs(b,a,$\omega$)，则不绘图输出，而是计算采样点 $\omega$ 处的频响并传递到 H 中。

**例 4.8 (原书习题 4-39 )** 若 $H(s)$ 零、极点分布如图 *4.5* 所示，试讨论它们分别是哪种滤波网络（低通、高通、带通、带阻）。

**解** Matlab 代码如下

```
>> data = struct('title',{'(a)','(b)','(c)',...      % 定义 data 为结构数组，
>>   '(d)','(e)','(f)','(g)','(h)'},'zeros',{[],... % 数组中每个元素都是结构，
```

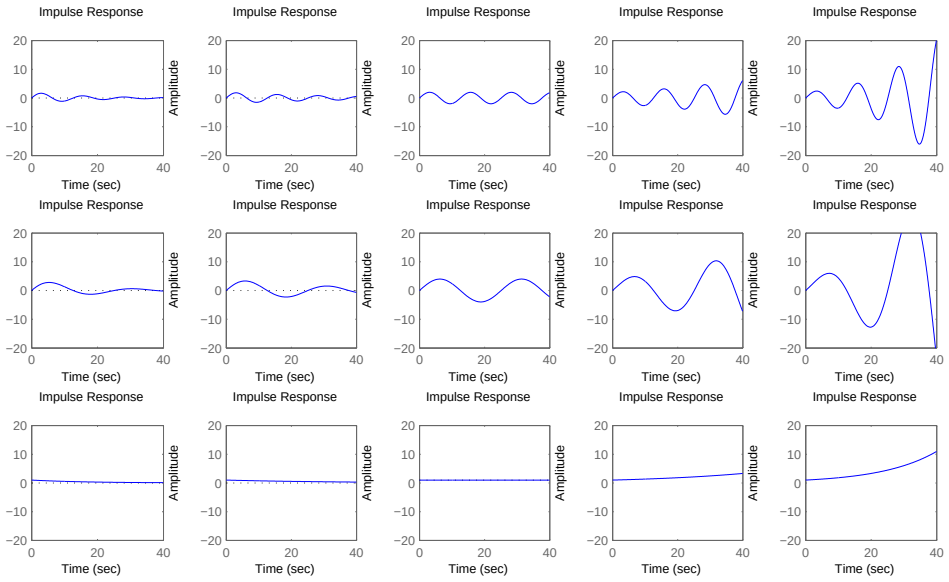


图 4.4: 例 4.7 运行结果

```
>>   [0],[0;0],[-0.5],[0],[1.2j;-1.2j],[0;0]},...      % 每个结构有三个域分别是
>>   [1.2j;-1.2j],'poles',{[-2;-1],[-2;-1],...         % 'title', 'zeros'和'poles'
>>   [-2;-1],[-2;-1],[-1+j;-1-j],[-1+j;-1-j],...
>>   [-1+j;-1-j],[j;-j]});
>> omega = [0:0.01:6];                                % 生成频率采样点
>> figure;                                            % 生成新图框
>> for id = 1:8                                       % 依次绘制 8 个系统的频响
>>   [b,a] = zp2tf(data(id).zeros,data(id).poles,1);  % 由零极点得到传递函数系数
>>   H = freqs(b,a,omega);                            % 计算指定频率点的响应 H(ω)
>>   subplot(4,2,id);                                 % 生成子图
>>   plot(omega,abs(H));                              % 绘制频响
>>   set(gca,'YScale','log','FontSize',16);           % 将 Y 轴设成对数座标，16 号字体
>>   title(data(id).title);                           % 绘制标题
>>   xlabel('\omega');                                % 添加横轴文本
>>   ylabel('H(\omega)');                             % 添加纵轴文本
>> end                                                % 循环结束
```

输出结果如图 4.6 所示，可见八个系统分别是：低通、带通、高通、带通、带通、带阻、高通和带通－带阻。

## Matlab 知识－结构

类似于 C 语言中的结构类型，Matlab 中的结构变量也是一种可以把不同类型的数据组合在一起的数据类型；和 C++ 的类一样，Matlab 中用操作符"."引用结构的成员变量（称作"域"）。用 struct 函数定义结构，比如

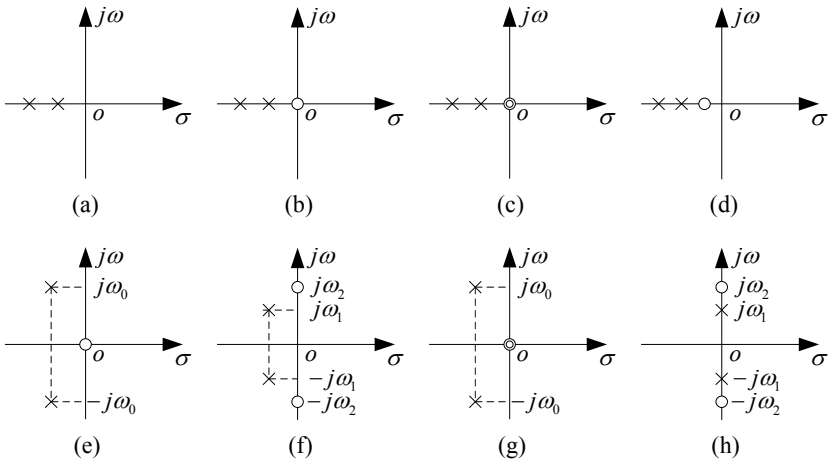


图 4.5: 例 4.8 系统零极点分布图

```
>> stu=struct('name',{'Tom'},'age',{6})
```

将创建一个包括两个域“name”和“age”的结构变量 stu，并且将这两个域分别初始化为“Tom”和 6，接下来就可以用“stu.name”和“stu.age” 分别引用这两个域。结构数组用类似的方法创建，比如

```
>> stus=struct('name',{'Tom','Jerry'},'age',{6,5})
```

创建了一个包括两个元素的结构数组“stus”，引用第二个元素的域的方式为“stus(2).name”。

如果对结构初始化时创建的域不足，还可以随时补充，比如就上例中，如果再输入

```
>> stus(1).gender='male'
```

则此结构除了“name”和“age”外，又增加了一个域“gender”，并且为“name”为“tom”的元素的“gender”域赋值为“male”，而另一个“name”为“Jerry”的元素的“gender”域则为空（“[]”）。

Matlab 还提供了诸如 isstruct 和 isfield 等函数实现判断结构和域的属性等各种基本操作，从而保证利用结构可以实现复杂的操作管理数据功能。

---

## 第五节　二阶谐振系统的 s 平面分析

根据原书公式 (4-119) 和 (4-122)，将二阶谐振系统写成标准形式：

$$Z(s) = \frac{s}{s^2 + 2\alpha s + \omega_0^2}$$

下面例题由 $\alpha$ 和 $\omega_0$ 的相对关系考察其时频特性。

**例 4.9 (原书图 4-37 修改)** 一个并联谐振电路，极点分布随 $\alpha$ 增大由虚轴上延单位圆旋转到负实轴，绘制电路系统的零极点图、冲激响应、幅频响应和相频响应。

**解** 我们将设计一个动画程序，希望程序运行后动态演示一对共轭极点从虚轴旋转到负实轴的过程中的零极点图、冲激响应和频率响应的变化。

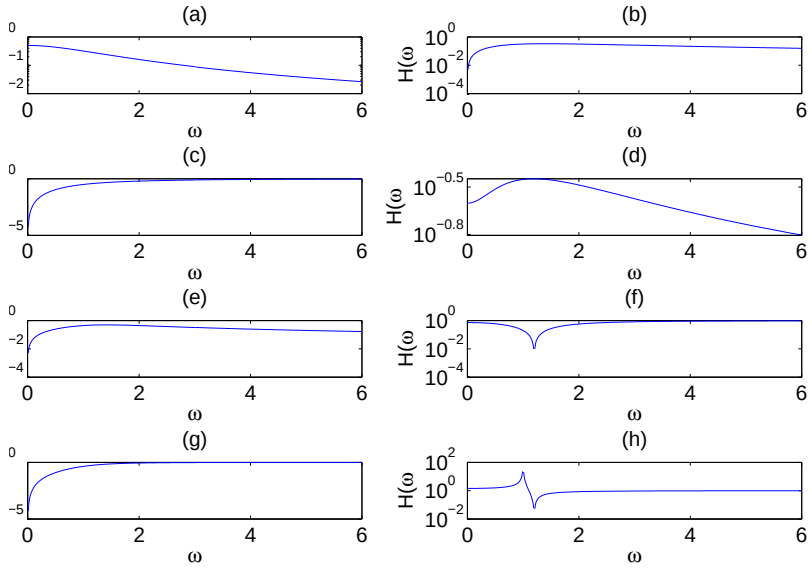


图 4.6: 例 4.8 运行结果

```
>> omega0 = 1;                                  % 定义 ω0 = 1
>> b = [1,0];                                   % 传递函数分子多项式系数不变
>> for theta = pi/2:pi/20:pi                    % 循环，θ 从 π/2 逐渐增大到 π
>>   alpha = omega0 * -cos(theta);              % 计算 α
>>   a = [1,2*alpha,omega0^2];                  % 定义传递函数分母多项式系数
>>   figure(1);                                 % 生成（激活）图框 1
>>   subplot(2,1,1);                            % 生成左上角的子图框
>>   pzmap(b,a);                                % 绘制零极点图
>>   set(gca,'XLim',[-1,1],'YLim',[-1,1]);      % 设置图框视野
>>   subplot(2,1,2);                            % 绘制左下角的子图框
>>   impulse(b,a);                              % 绘制冲激响应
>>   figure(2);                                 % 生成（激活）图框 2
>>   freqs(b,a);                                % 绘制幅度和相位的频率响应
>>   pause(0.2);                                % 暂停，迫使绘图事件发生
>> end                                          % 循环结束
```

图4.7(a) 和 (d) 分别绘制出极点位于虚轴时的情况，图4.7(b) 、(e) 、(c) 和 (f) 分别绘制出极点位于第二、三象限和负实轴上的情况。

上例中的 pause 语句用于产生暂停效果，以便绘图函数执行结束并切实把绘图效果在屏幕上显示出来。

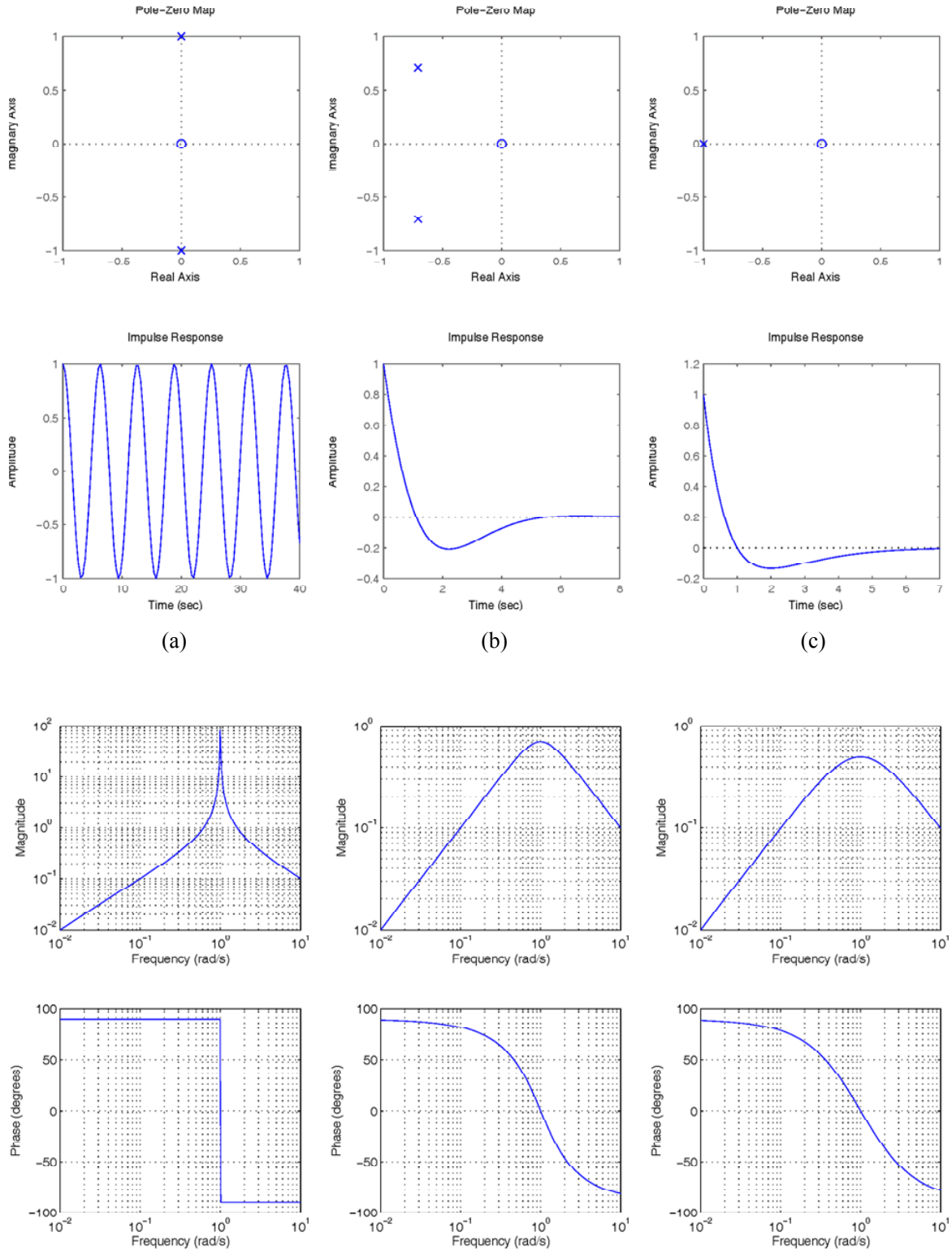


图 4.7: 例 4.9 程序运行结果

# 第五章　音乐合成实验

本章中将基于傅里叶级数和傅里叶变换等《信号与系统》讲授的基本内容，应用前三章介绍的 Matlab 编程技术，在电子音乐合成领域做一些练习。通过本章的练习，希望同学们可以增进对傅里叶级数的理解，熟练运用 Matlab 基本指令。本章包括两部分，第一部分介绍乐理和电子音乐的基本知识，第二部分给出详细的练习内容和编程步骤。

## 第一节　背景知识

### 5.1.1　音乐特征

人类听觉可以感受到的声音大体上可划分为噪声、语音、乐音……等几种类型。关于噪声和语音的特征及其性能分析将在本科高年级或研究生的课程中专门研究。本练习初步介绍乐音的基本概念，并利用 Matlab 编程实现音乐合成系统。

音乐是乐音随时间流动而形成的艺术。用通信与电子技术的术语解释就是周期信号频率（某种指定规律的频谱结构）随时间节奏变化的一种表述。乐谱上的每个音符表达了此时此刻规定出现的信号频率和持续时间。

乐音的基本特征可以用基波频率、谐波频谱和包络波形三个方面来描述，下面将分别说明。

很明显，认识乐音的频谱规律之后，我们就可以借助电子系统从软件或硬件两种角度模仿各种乐器产生的声音，实习所谓电子音乐系统。

电子音乐系统是一门新兴的交叉技术科学，涉及计算机、集成电路、电子线路、信号处理、声学等多种领域，研究与应用前景广阔而深远。本练习只是非常简单的入门介绍，相信同学们会产生很大的兴趣。

### 5.1.2　音乐基波构成规律

我们用 $CDEFGAB$ 大写英文字母表示每个音的“音名”（或称为“音调”），当指定某一音名时，它对应固定的基波信号频率。

图 5.1 示出钢琴键盘结构，并注明了每个琴键对应的音名和基波频率值。这些频率值是按“十二平均律”计算导出，下面解释计算规则。

从图 5.1 可以看到，靠下边的 $A$ 键称为小字组 $A$，它的频率值 $f_{A0} = 220\text{Hz}$。而靠上面的另一个 $A$ 键是小字一组 $A$，它的频率值是 $f_{A1} = 440\text{Hz}$。两者为二倍频率关系，即 $f_{A1}$ 相当于 $f_{A0}$ 的二次谐波。也称为 8 度音程或倍频程 Octave（即我们画频响特性波特图时所用的术语“倍频程”）。

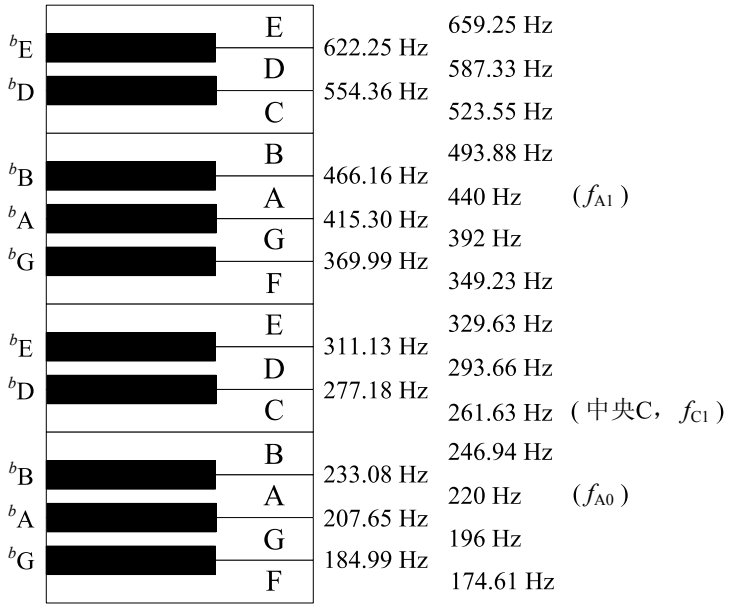


图 5.1: 钢琴键盘和相应频率

从小字组 $A$ 到小字一组 $A$ 之间共有 12 个键，其中 7 个白色键，5 个黑色键，其频率值计算规律为相邻音倍乘系数 $K = 2^{\frac{1}{12}} = 1.05946309$ 。

由此可求出图中各琴键对应之频率值。例如从 $f_{A0}$ 导出小字一组 $C$（也称为中央 $C$ ）的频率为

$$f_{C1} = 220 \times 2^{\frac{3}{12}}\text{Hz} = 261.63\text{Hz} \tag{5.1}$$

或利用 $f_{A1}$ 也可导出同样结果。

人耳听觉辨识振动频率的能力大约在 $\pm 0.5\%$ ，上述规律刚好符合这一要求。

从图 5.1 可以看出 7 个白键之间插入了 5 个黑键。在 $EF$ 之间和 $BC$ 之间没有黑键，也即这两组相邻的白键之间基波频率倍乘系数为 $2^{\frac{1}{12}}$ ，也称为相隔半音，而在其他白键之间都有黑键相隔，因而他们的频率倍乘系数为 $2^{\frac{2}{12}}$ ，也称为相隔全音（如 $CD$ 、$DE$ ， $FG$ 、……之间）。若以白键英文字母为基准，则升高半音以“ ♯ ”符号表示，降低半音则以“$b$”符号表示。于是，可以写出 12 个音名从低到高的依次字母表示为

$$C,\ {}^{b}D,\ D,\ {}^{b}E,\ E,\ F,\ {}^{b}G,\ G,\ {}^{b}A,\ A,\ {}^{b}B,\ C \tag{5.2}$$

当然，若改用“ ♯ ”号表示黑键，则 ${}^{b}D$ 改为 ${}^{\sharp}C$ ， ${}^{b}E$ 改为 ${}^{\sharp}D$ ，……等等。

下面给出“唱名”的概念。所谓唱名是指平日读乐谱唱出的 do 、re 、mi 、……。每个唱名并未固定基波频率。当指定乐曲的音调时才知道此时唱名对应的音名，也即确定了对应的频率值。

例如，若指定乐曲为 $C$ 调（或称 $C$ 大调），此时唱名与音名的对应关系如图 5.2(a) 。图中将唱名 do 、re 、mi 、……以简谱符号 1 、2 、3 、……代替。可见，它的“1”对应“ $C$ ”也即基波频率为 261.63Hz 。全部唱名对应键盘的白键。还可看出，3 、4 分别对应 $E$ 、$F$ 二者之间是半音阶，7 、i 之间对应 $B$ 、$C$ 也属半音。其它唱名之间都为全音。

如果改为 $F$ 调（ $F$ 大调），唱名与音名的对应关系如图 5.2(b) 。它的“1”与 $F$ 对应，频率值是 349.23Hz 。为了保持 3 、4 之间为半音之规律，“ 4 ”对应黑键 ${}^{b}B$ 。

为了保持 3 、4 以及 7 、i 之间为半音之规律，只有 $C$ 调的全部唱名都与白键对应。而其它各调都需要引用黑键。如上例 $F$ 调的“ 4 ”对应 ${}^{b}B$ ，其它各调对应黑键的规律可作为练习请读者自行导出。


(a) C调唱名与音名之对应　　(b) F调唱名与音名之对应

图 5.2: 唱名与音名之对应

$$1 = F\,\frac{2}{4} \quad | \; 5 \quad \overset{\frown}{5\,6} \; | \; 2 \; -- \; | \; 1 \quad \overset{\frown}{1\,\dot{6}} \; | \; 2 \quad -- \; |$$

图 5.3: 乐曲“东方红”第一小节曲谱

下面给出一个乐曲实例，练习写出每个唱名对应的基波频率值，如图 5.3 。这是“东方红”的开头两句曲谱，用简谱写出。曲调定为 $F$ ，即 $1 = F$ ，于是可查出第一个音 5 对应 $C$ ，频率值为 523.55Hz ，其它音之频率可依次写出。稍后，我们将以此为基础进行音乐合成练习。

### 5.1.3　乐音谐波的作用－音色

当指定音名（音调）之后仅指定了乐音信号的基波频率，谐波情况并未说明。对于各种乐器如钢琴或单簧管都可发出 $f_{A1} = 440$Hz 之乐音，而人的听觉会明显感觉二者不同，这是由于谐波成分有所区别，频谱结构各异。例如单簧管的三次、五次谐波成分很强，其它各种乐器都有自己的谐波分布规律。同种乐器不同音阶之谐波构成还可能略有区别。由于演奏技巧、方法之差异也可产生不同结构之谐波。

在制作电子乐器（如电子琴）时，应尽力模仿实际乐器之谐波结构。但是由于人为因素的随机变化，往往感觉电子琴产生的乐音与利用传统乐器产生的乐音很难完全一致。

在音乐领域中称谐波为“泛音”。由谐波产生的作用称为音色变化。

在稍后的编程练习中将要看到，如果只考虑乐音的基波成分，每个音名对应不同频率的正弦（余弦）波；当引入谐波分量之后，波形不再是简单的正弦函数，例如，可能接近矩形、锯齿波、……等等。

### 5.1.4　乐音波形包络

这是描述乐音特性的另一个重要因素。除了前述基频和谐波表征了乐音特性之外，对于不同类型的乐器它们的包络形状也不相同，在电子乐器制作中习惯上称此包络为“音型”或“音形”。实际的波形好像通信系统中经过调制的信号。

各种乐器的包络（音型）大体上可划分为以下几种类型：

- 连续型（风琴、手风琴、弦乐）
- 弹奏型（钢琴、吉他）
- 拨奏型（琵琶、月琴、曼多林）
- 击奏型（木琴、木鱼）
- 吹奏型（管乐）

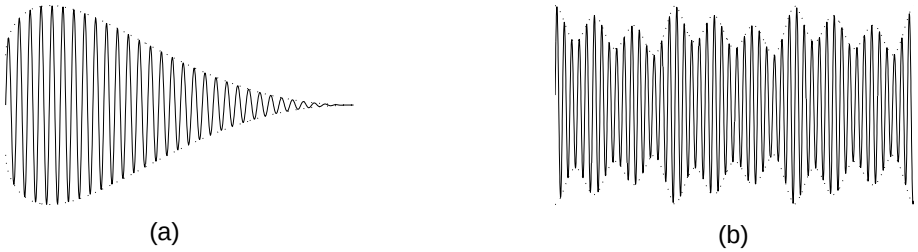


图 5.4: 钢琴和管乐的信号波形

- 颤音型（调频信号，如小提琴揉音）

图 5.4 示出钢琴与管乐的波形，可以看出二者包络之区别。其它类型省略。

在乐音合成实验中，为简化编程描述，通常把复杂的包络函数用少量直线近似，于是，乐音波形之包络呈折线。

### 5.1.5　音调持续时间

在我们最简单的例子中，每个音调都可以用连续的一段正弦信号并带有一小段静音（停顿）来表示。停顿保证我们可以区分开连续的相同音调。每个音调的持续时间取决于它是全音符、二分之一音符、四分之一音符还是八分之一音符等等。明显的，四分之一音符的持续时间是八分之一音符的两倍。而每个音符之后的停顿时间应该是相同的，不随音符的长度而变化。在乐谱中，更长一些的停顿要用休止符来表示。图 5.3 中左侧的两个 4 表示每小节有四拍和一个四分之一音符持续一拍。对这段乐曲来说，一拍大约是 1/2 秒。

## 第二节　练习题

### 5.2.1　最简单的合成音乐

(1) 请根据“东方红”第一小节的简谱和“十二平均律”计算出该小节中各个乐音的频率，在 Matlab 中生成幅度为 1 采样频率为 8kHz 的正弦信号表示这些乐音。请用 sound 函数播放每个乐音，听一听音调是否正确。最后用这一系列乐音信号拼出“东方红”第一小节，注意控制每个乐音持续的时间要符合节拍，用 sound 播放你合成的音乐，听起来感觉如何？

(2) 你一定注意到 (1) 的乐曲中相邻乐音之间有“啪”的杂声，这是由于相位不连续产生了高频分量。这种噪声严重影响合成音乐的质量，丧失真实感。为了消除它，我们可以用图 5.5 所示包络修正每个乐音，以保证在乐音的邻接处信号幅度为零。最简单的，也可以用指数衰减的包络来表示。建议：为了让听觉感受为线性变化，功率应该呈指数变化。

(3) 请用最简单的方法将 (2) 中的音乐分别升高和降低一个八度。（提示：音乐播放的时间可以变化）再难一些，请用 resample 函数（也可以用 interp 和 decimate 函数）将上述音乐升高半个音阶。（提示：视计算复杂度，不必特别精确）

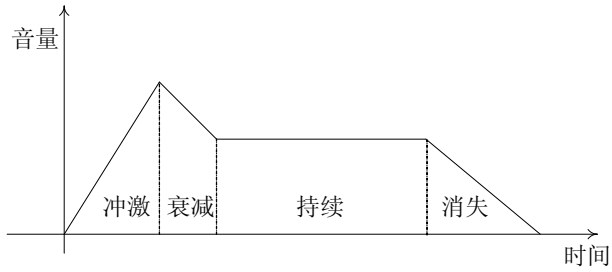


图 5.5: 音量变化

(4) 试着在 (2) 的音乐中增加一些谐波分量，听一听音乐是否更有“厚度”了？注意谐波分量的能量要小，否则掩盖住基音反而听不清音调了。（我选择基波幅度为 1 ，二次谐波幅度 0.2 ，三次谐波幅度 0.3 ，听起来有点象风琴。）

### 5.2.2　用傅里叶级数分析音乐

现在我们开始要处理真实的音乐信号了！请用 load 命令载入附件光盘中的数据文件“guitar.mat”，工作区会出现两个新的变量 realwave 和 wave2proc（可以用 who 查看变量名），如图 5.6 和 5.7 所示。其中前者是从一段吉他乐曲（附件光盘上的“fmt.wav”）中截取下来的真实信号，后者是用信号处理方法得到的这段信号的理论值，它们的采样率都是 8kHz 。

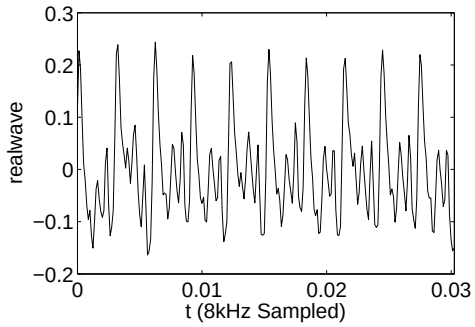


图 5.6: 真实吉他音

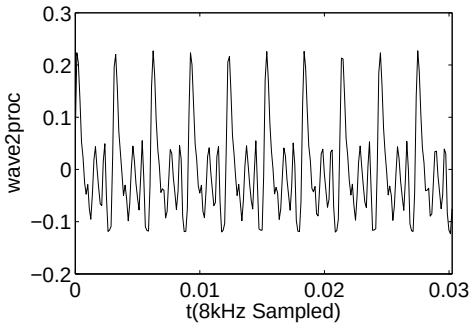


图 5.7: 待处理的吉他音

(5) 先用 wavread 函数载入光盘中的 fmt.wav 文件，播放出来听听效果如何？是否比刚才的合成音乐真实多了？

---

**Matlab 知识－访问标准格式文件**

为了充分发挥其强大的数据处理能力的影响，Matlab 支持对工业界多种标准格式文件的读写访问。表 5.1 中列出了一些常用的函数。

为了支持用户在程序中保存 Figure 窗口中的图形图像，Matlab 提供了 saveas 函数，和用户在窗口中点击“File→Save As”一样，它支持的包括 pdf 在内的图像文件格式有 16 种之多！

---

(6) 你知道待处理的 wave2proc 是如何从真实值 realwave 中得到的么？这个预处理过程可以去除真实乐曲中的非线性谐波和噪声，对于正确分析音调是非常重要的。提示：从时域做，可以继续使用 resample 函数。

表 5.1: 访问标准格式文件的常用函数

| 函数名 | 说明 |
|---|---|
| dlmread , dlmwrite , textscan , dlmread , dlmwrite , csvread , csvwrite , 等 | 访问各种 ASCII 文本格式文件 |
| xlsread , xlswrite | 访问微软 EXCEL 格式文件 |
| wavread , wavwrite | 访问微软 WAV 格式音频文件 |
| auread , auwrite | 访问 NeXT/SUN 格式音频文件 |
| imread , imwrite | 访问 JPEG, TIFF, GIF, BMP, PNG 等多种格式图像文件 |
| aviread , avifile , 等 | 访问 AVI 视频文件 |

(7) 这段音乐的基频是多少？是哪个音调？请用傅里叶级数或者变换的方法分析它的谐波分量分别是什么？提示：简单的方法是近似取出一个周期求傅里叶级数但这样明显不准确，因为你应该已经发现基音周期不是整数（这里不允许使用 resample 函数）。复杂些的方法是对整个信号求傅里叶变换（回忆周期性信号的傅里叶变换），但你可能发现无论你如何提高频域的分辨率，也得不到精确的包络（应该近似于冲激函数而不是 sinc 函数），可选的方法是增加时域的数据量，即再把时域信号重复若干次，看看这样是否效果好多了？请解释之。

(8) 再次载入 fmt.wav ，现在要求你写一段程序，自动分析出这段乐曲的音调和节拍！如果你觉得太难就允许手工标定出每个音调的起止时间，再不行你就把每个音调的数据都单独保存成一个文件，然后让 Matlab 对这些文件进行批处理。注意：不允许逐一地手工分析音调。编辑音乐文件，推荐使用“CoolEdit” 编辑软件。

### 5.2.3 基于傅里叶级数的合成音乐

现在进入了合成音乐的高级境界，我们要用演奏 fmt.wav 的吉他合成出一段“东方红”。

(9) 用 (7) 计算出来的傅里叶级数再次完成第 (4) 题，听一听是否象演奏 fmt.wav 的吉他演奏出来的？

(10) 也许(9) 还不是很像，因为对于一把泛音丰富的吉他而言，不可能每个音调对应的泛音数量和幅度都相同。但是通过完成第 (8) 题，你已经提取出 fmt.wav 中的很多音调，或者说，掌握了每个音调对应傅里叶级数，大致了解了这把吉他的特征。现在就来演奏一曲“东方红”吧。提示：如果还是音调信息不够，那就利用相邻音调的信息近似好了，毕竟可以假设吉他的频响是连续变化的。

(11) 现在只要你掌握了某乐器足够多的演奏资料，就可以合成出该乐器演奏的任何音乐，再学完本书后面内容之后，试着做一个图形界面把上述能力封装起来。

# 本书所用到的 Matlab 命令、说明及页码