



北京理工大学  
Beijing Institute of Technology

# 本科生实验报告

课程名称: 信号与线性系统

实验名称: 实验 3 傅里叶变换的编程实现

任课教师:	[REDACTED]		实验教师:	[REDACTED]
实验日期:	[REDACTED]		实验地点:	[REDACTED]
实验类型:	<input checked="" type="checkbox"/> 原理验证 <input checked="" type="checkbox"/> 综合设计 <input checked="" type="checkbox"/> 自主创新			
学生姓名:	[REDACTED]	班级:	[REDACTED]	学号: [REDACTED]
学院:	信息与电子学院		专业:	电子信息工程（徐特立英才班）
组号:	[REDACTED]	同组同学:	(无需填写)	
成绩:				



信息与电子学院

SCHOOL OF INFORMATION AND ELECTRONICS

# 实验 3 傅里叶变换的编程实现

## 一、 实验目的

1. 掌握傅里叶变化的数值计算方法。
2. 掌握 MATLAB 程序优化的技巧, 对比傅里叶变换矢量实现和矩阵实现的运行效率。

## 二、 实验原理

1. 工程实现中处理的是离散数据, 在这种情况下往往无法得到信号的解析表达式, 数值计算方法称为傅里叶变换实现的主要途径。傅里叶变换的表达式为

$$F(\omega) = \int_{-\infty}^{+\infty} f(t) e^{-j\omega t} dt = \int_{t_2}^{t_1} f(t) e^{-j\omega t} dt \quad (1)$$

其中  $[t_1, t_2]$  为主值区间, 定义  $T$  为数据长度,  $N$  为采样点数, 采样间隔为  $\Delta t = T/N$ , 得到:

$$F(\omega) = \frac{T}{N} \sum_{n=1}^{N-1} f(t_1 + n\Delta t) e^{-j\omega(t_1+n\Delta t)} \quad (2)$$

假设  $F(\omega)$  的取值区间为  $[\omega_1, \omega_2]$ , 计算任意一个频域采样点  $k$  的值, 则有

$$F(\omega_1 + k\Delta\omega) = \frac{T}{N} \sum_{n=0}^{N-1} f(t_1 + n\Delta t) e^{-j(\omega_1+k\Delta\omega)(t_1+n\Delta t)} \quad (3)$$

### 2. 矢量计算法

上式的累加可以用内积来实现, 即

$$\begin{aligned} F(\omega_1 + k\Delta\omega) &= \frac{T}{N} [e^{-j(\omega_1+k\Delta\omega)t_1} e^{-j(\omega_1+k\Delta\omega)(t_1+\Delta t)} \dots \\ &\quad e^{-j(\omega_1+k\Delta\omega)(t_1+(N-1)\Delta t)}] \begin{bmatrix} f(t_1) \\ \vdots \\ f(t_1 + (N-1)\Delta t) \end{bmatrix} \end{aligned} \quad (4)$$

同理可得出逆傅里叶变换的矢量运算

$$f(t_1 + n\Delta t) = \frac{\Omega}{2\pi K} [e^{j\omega_1(t_1+n\Delta t)} e^{j(\omega_1+\Delta\omega)(t_1+n\Delta t)} \dots e^{j(\omega_1+(K-1)\Delta\omega)(t_1+n\Delta t)}] \begin{bmatrix} F(\omega_1) \\ F(\omega_1 + \Delta\omega) \\ \vdots \\ \vdots \\ F(\omega_1 + (K-1)\Delta\omega) \end{bmatrix} \quad (5)$$

### 3. 矩阵计算法

公式(4)还可以写成矩阵形式:

$$\begin{bmatrix} F(\omega_1) \\ F(\omega_1 + \Delta\omega) \\ \vdots \\ \vdots \\ F(\omega_1 + (K-1)\Delta\omega) \end{bmatrix} = \frac{T}{N} \begin{bmatrix} e^{-j\omega_1 t_1} & e^{-j\omega_1(t_1+n\Delta t)} & \dots & e^{-j\omega_1(t_1+(N-1)\Delta t)} \\ e^{-j(\omega_1+\Delta\omega)t_1} & e^{-j(\omega_1+\Delta\omega)(t_1+n\Delta t)} & \dots & e^{-j(\omega_1+\Delta\omega)(t_1+(N-1)\Delta t)} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ e^{-j(\omega_1+(K-1)\Delta\omega)t_1} & e^{-j(\omega_1+(K-1)\Delta\omega)(t_1+n\Delta t)} & \dots & e^{-j(\omega_1+(K-1)\Delta\omega)(t_1+(N-1)\Delta t)} \end{bmatrix} \begin{bmatrix} f(t_1) \\ f(t_1 + \Delta t) \\ \vdots \\ \vdots \\ f(t_1 + (N-1)\Delta t) \end{bmatrix} \quad (6)$$

可简写为:

$$F = \frac{T}{N} U f \quad (7)$$

## 三、实验内容

- 分别利用矢量计算方法实现实验一中周期矩形脉冲的傅里叶变换和逆傅里叶变换;

MATLAB 代码如下:

```

1. %% 矢量计算方法
2. % 参数设置
3. T0 = 2;
4. tau = 0.5;
5. Fs = 1000;
6. T = 1/Fs;
7. t_total = 10*T0;
8. t = -t_total/2 : T : t_total/2 - T;
9. N = length(t);
10.
11. % 生成周期矩形脉冲信号
12. x = zeros(size(t));

```

```
13. for n = -round(t_total/(2*T0)) : round(t_total/(2*T0))
14.     x = x + (abs(t - n*T0) < tau/2);
15. end
16.
17. % 确定主值区间
18. t1 = min(t);
19. t2 = max(t);
20. T_data = t2 - t1;
21.
22. % 频率参数设置
23. f_center = 0;
24. f_span = Fs;
25. K = N;
26. df = f_span/N;
27.
28. % 构建对称频率轴
29. f = f_center - f_span/2 : df : f_center + f_span/2 - df;
30.
31. % 矢量计算方法 - 傅里叶变换
32. tic
33. X_vec = zeros(1, K);
34.
35. % 预计算时间序列
36. time_series = t1 + (0:N-1)*T; % t1, t1+Δt, ..., t1+(N-1)Δt
37.
38. % 计算 DTFT
39. for k = 1:K
40.     current_freq = f(k);
41.
42.     exponent = -1i * 2*pi * current_freq * time_series;
43.     exp_term = exp(exponent);
44.
45.     X_vec(k) = sum(x .* exp_term) * T;
46. end
47. toc
48.
49. % 矢量计算方法 - 逆傅里叶变换
50. tic
51. x_vec_ifft = zeros(1, N);
52.
53. % 预计算频率序列
54. freq_series = f;
55.
56. % 计算 IDTFT
```

```

57. for n = 1:N
58.     current_time = t1 + (n-1)*T; % 当前时间点
59.     exponent = 1i * 2*pi * freq_series * current_time;
60.     exp_term = exp(exponent);
61.     x_vec_ifft(n) = sum(X_vec .* exp_term) * df; % 乘以频率间隔 df
62. end
63. x_vec_ifft = real(x_vec_ifft);
64. toc
65.
66. % 绘图显示
67. figure;
68. subplot(3,1,1);
69. plot(t, x);
70. title('原始周期矩形脉冲');
71. xlabel('时间 (s)');
72. ylabel('幅度');
73. grid on;
74.
75. subplot(3,1,2);
76. plot(f, abs(X_vec));
77. title('矢量计算 - 傅里叶变换幅度谱');
78. xlabel('频率 (Hz)');
79. ylabel('|X(f)|');
80. grid on;
81.
82. subplot(3,1,3);
83. plot(t, x_vec_ifft);
84. title('矢量计算 - 逆傅里叶变换恢复信号');
85. xlabel('时间 (s)');
86. ylabel('幅度');
87. grid on;

```

本次实验中参数选取与实验一相同，均取  $T_0 = 2$ ,  $\tau = 0.5$ ,  $A = 1$ ，并选取对称于零时刻的十个周期，利用矢量计算得到的傅里叶变换频谱图如下，并且通过傅里叶逆变换可以恢复原信号。

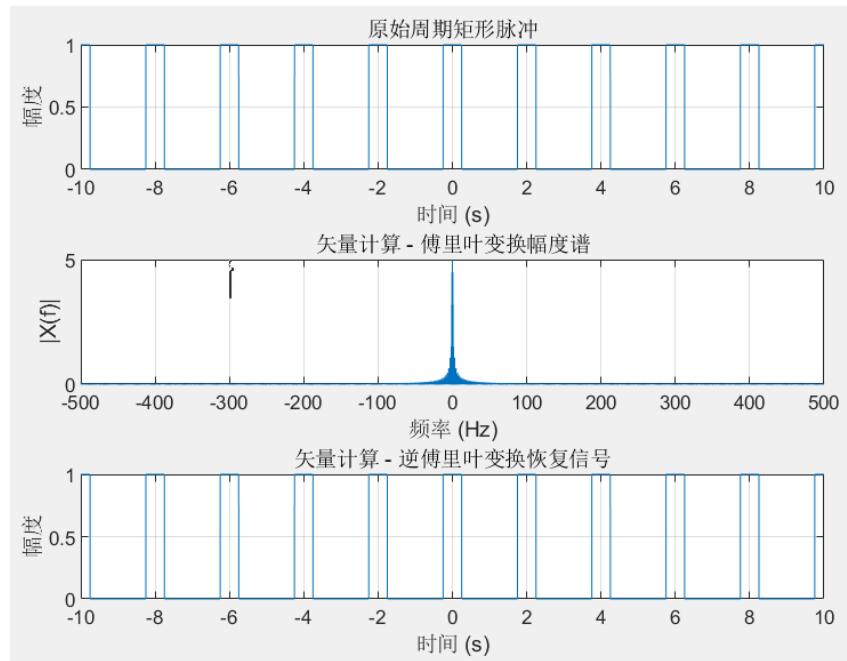


图 1：矢量计算傅里叶变换频谱图及恢复信号

观察可知，矢量计算方法得到的频谱图和实验一中的结果非常相似，并且经过傅里叶逆变换可以很好地恢复时域信号，验证了本次实验的正确性。

## 2. 分别利用矩阵计算方法实现实验一中周期矩形脉冲的傅里叶变换和逆傅里叶变换；

MATLAB 代码如下：

```

1. %% 矩阵计算方法（直接接在第一问代码后面）
2. % 矩阵计算方法 - 傅里叶变换
3. tic
4. % 构建时间矩阵（N 行 K 列）
5. t_matrix = ones(K, 1) * time_series;
6. % 构建频率矩阵（N 行 K 列）
7. f_matrix = freq_series' * ones(1, N);
8.
9. % 构建指数矩阵 U
10. exponent_matrix = -1i * 2*pi * f_matrix .* t_matrix;
11. U = exp(exponent_matrix);
12.
13. % 计算傅里叶变换
14. X_mat = x * U * T;
15. toc
16.
17. % 矩阵计算方法 - 逆傅里叶变换
18. tic
19. % 构建逆变换指数矩阵

```

```

20. U_inv = exp(1i * 2*pi * f_matrix .* t_matrix);
21. % 计算逆傅里叶变换
22. x_mat_ifft = X_mat * U_inv * df;
23. x_mat_ifft = real(x_mat_ifft);
24. toc
25.
26. % 绘图显示
27. figure;
28. subplot(3, 1, 1);
29. plot(t, x);
30. title('周期矩形脉冲信号');
31. xlabel('时间 (s)');
32. ylabel('幅度');
33. grid on;
34. subplot(3, 1, 2);
35. plot(f, abs(X_mat));
36. title('矩阵计算 - 傅里叶变换幅度谱');
37. xlabel('频率 (Hz)');
38. ylabel('|X(f)|');
39. grid on;
40. subplot(3, 1, 3);
41. plot(t, x_mat_ifft);
42. title('矩阵计算 - 逆傅里叶变换恢复信号');
43. xlabel('时间 (s)');
44. ylabel('幅度');
45. grid on;

```

这里参数选取与上一问相同，得到的结果如下：

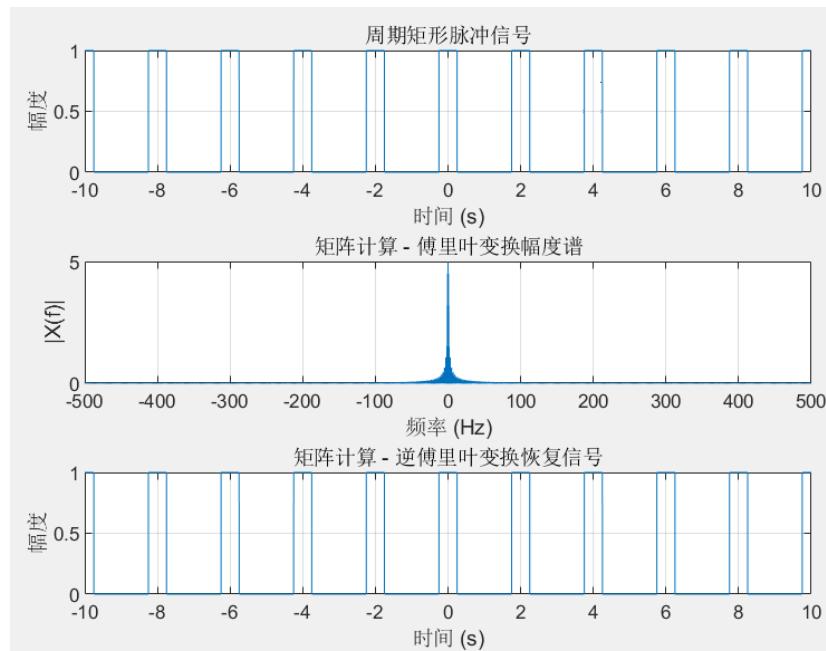


图 2：矩阵计算傅里叶变换频谱图及恢复信号

观察可知，矩阵计算方法得到的频谱图与上一节和实验一中的结果非常相似，并且恢复信号的波形与原信号相同，这进一步验证了我们算法的正确性。

### 3. 利用 **tic** 和 **toc** 指令比较两种方法的运算效率。

在运行过程中，利用矢量计算方法进行傅里叶变换用时 3.460580s，进行逆傅里叶变换用时 4.716443s。利用矩阵计算方法进行傅里叶变换用时 25.959040s，进行逆傅里叶变换用时历时 23.594357s。

由上述比较可以看出：矢量计算方法运行时间显著短于矩阵计算方法。在计算过程中矩阵计算方法需要构建并存储一个庞大的指数矩阵，从而导致频繁的高延迟主存访问和内存分配开销；而矢量计算方法通过逐点循环计算，仅需维护一维向量，显著减少内存占用，能更有效利用 CPU 缓存，加之 MATLAB 的循环优化机制，从而大幅降低实际运行时间。