

安徽大学人工智能学院

《数字信号处理》

实验案例设计报告

课程名称: 数字信号处理实验

专 业: 人工智能

班 级: 人工智能二班

学 号: WA2214014

姓 名: 杨跃浙

任课老师: 谭春雨

实验名称	实验四	实验次序	04
实验地点	笃行南楼 A301	实验日期	06.10
<p>实验内容:</p> <p>8-1</p> <p>实验目的</p> <p>通过编程实现离散傅里叶变换 (DIT-FFT) , 计算给定序列 $x(n)$ 的频域表示 $X(k)$。通过对比手动实现的 FFT 结果和 MATLAB 内置 FFT 函数的结果, 验证手动实现的正确性和有效性, 加深对 FFT 算法的理解。</p> <p>实验原理</p> <p>离散傅里叶变换 (DFT) 是一种将离散时间序列转换为频域表示的方法。快速傅里叶变换 (FFT) 是一种高效计算 DFT 的算法。蝶形运算 (DIT-FFT) 是一种基于时间抽取的 FFT 算法。算法的核心思想是利用分治法, 将长度为 N 的 DFT 分解为若干个长度为 2 的小 DFT, 通过蝶形运算逐步合并计算结果。</p> <p>实验代码</p> <p>函数 ditfft ()</p> <pre> function Xk = ditfft(xn) M = nextpow2(length(xn)); N = 2^M; for m = 0:N/2-1 WN(m+1) = exp(-j * 2 * pi * m / N); end A = [xn, zeros(1, N - length(xn))]; disp('输入到各存储单元的数据:'); disp(A); J = 0; for I = 0:N-1 if I < J T = A(I+1); </pre>			

```

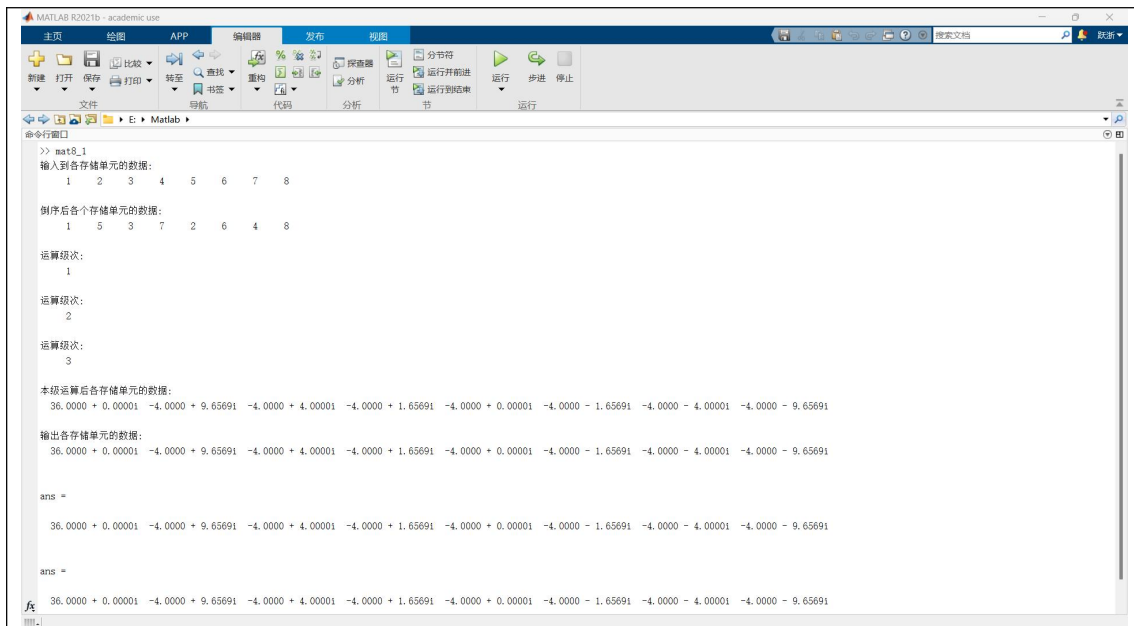
A(I+1) = A(J+1);
A(J+1) = T;
end
K = N / 2;
while J >= K
J = J - K;
K = K / 2;
end
J = J + K;
end
disp('倒序后各个存储单元的数据:');
disp(A);

for L = 1:M
disp('运算级次:');
disp(L);
B = 2^(L-1);
for R = 0:B-1
P = 2^(M-L) * R;
for K = R:2^L:N-2
T = A(K+1) + A(K+B+1) * WN(P+1);
A(K+B+1) = A(K+1) - A(K+B+1) * WN(P+1);
A(K+1) = T;
end
end
end
disp('本级运算后各存储单元的数据:');
disp(A);
Xk = A;
disp('输出各存储单元的数据:');
disp(Xk);
end

xn=[1,2,3,4,5,6,7,8];
ditfft(xn)
fft(xn)

```

实验结果



```
>> mat8_1
输入到各存储单元的数据:
1 2 3 4 5 6 7 8

倒序后各存储单元的数据:
1 5 3 7 2 6 4 8

运算级次:
1

运算级次:
2

运算级次:
3

本级运算后各存储单元的数据:
36.0000 + 0.0000i -4.0000 + 9.6569i -4.0000 + 4.0000i -4.0000 + 1.6569i -4.0000 + 0.0000i -4.0000 - 1.6569i -4.0000 - 4.0000i -4.0000 - 9.6569i

输出各存储单元的数据:
36.0000 + 0.0000i -4.0000 + 9.6569i -4.0000 + 4.0000i -4.0000 + 1.6569i -4.0000 + 0.0000i -4.0000 - 1.6569i -4.0000 - 4.0000i -4.0000 - 9.6569i

ans =

36.0000 + 0.0000i -4.0000 + 9.6569i -4.0000 + 4.0000i -4.0000 + 1.6569i -4.0000 + 0.0000i -4.0000 - 1.6569i -4.0000 - 4.0000i -4.0000 - 9.6569i

ans =

36.0000 + 0.0000i -4.0000 + 9.6569i -4.0000 + 4.0000i -4.0000 + 1.6569i -4.0000 + 0.0000i -4.0000 - 1.6569i -4.0000 - 4.0000i -4.0000 - 9.6569i
```

8-2

实验目的

通过编程实现逆离散傅里叶变换 (IDFT) , 计算给定频谱 $X(k)$ 对应的时间序列 $x(n)$ 。通过对比手动实现的 IDFT 结果和 MATLAB 内置 `ifft` 函数的结果, 验证手动实现的正确性和有效性, 加深对 IDFT 算法的理解。

实验原理

逆离散傅里叶变换 (IDFT) 用于将频域信号转换回时间域信号。利用傅里叶变换的共轭对称性, 可以通过对频谱 $X(k)$ 取共轭后进行快速傅里叶变换 (FFT), 再对结果取共轭并归一化, 得到对应的时间序列 $x(n)$ 。这样, 通过使用 FFT 和共轭运算, 可以高效地实现 IDFT。最终, 通过对比手动计算的 IDFT 结果和 MATLAB 内置的 `ifft` 函数的结果, 可以验证算法的正确性和效率。

实验代码

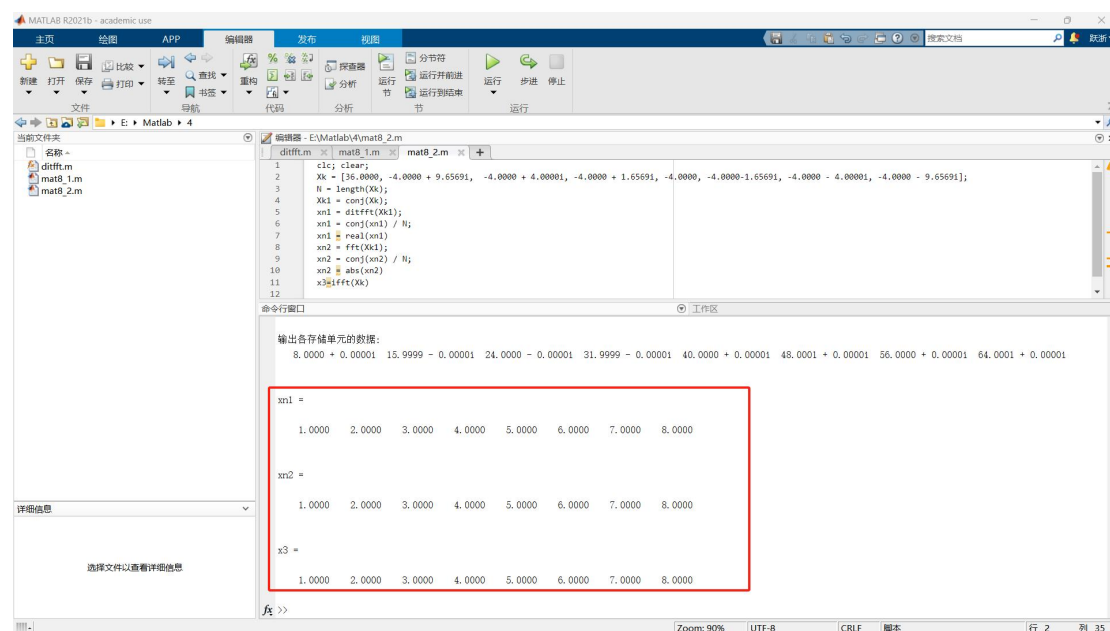
```
clc; clear;
Xk = [36.0000, -4.0000 + 9.6569i, -4.0000 + 4.0000i, -4.0000 + 1.6569i, -4.0000, -4.0000-1.6569i,
-4.0000 - 4.0000i, -4.0000 - 9.6569i];
```

```

N = length(Xk);
Xk1 = conj(Xk);
xn1 = ditfft(Xk1);
xn1 = conj(xn1) / N;
xn1 = real(xn1)
xn2 = fft(Xk1);
xn2 = conj(xn2) / N;
xn2 = abs(xn2)
x3=ifft(Xk)

```

实验结果



8-3

实验目的

通过对比直接计算离散傅里叶变换（DFT）和快速傅里叶变换（FFT）的执行时间，评估两种方法在计算效率上的差异。通过实验结果验证 FFT 在计算大规模 DFT 时的高效性，从而加深对 FFT 算法优越性的理解。

实验原理

离散傅里叶变换（DFT）是一种将时间域信号转换为频域信号的方法，其直

接计算需要 $O(N^2)$ 的复杂度，计算效率较低。快速傅里叶变换 (FFT) 是一种高效计算 DFT 的算法，其复杂度仅为 $O(N\log N)$ ，显著提高了计算速度。通过编程实现两种方法的 DFT 计算，并对比它们在不同序列长度下的执行时间，可以直观地展示 FFT 相对于直接计算 DFT 的时间优势。

在实验中，将对不同长度的随机序列分别使用直接计算 DFT (DFTfor 和 DFTmat) 和快速算法 FFT (DIT-FFT) 进行变换，并记录每种方法的执行时间。通过绘制执行时间与序列长度的关系图，可以清晰地对比两种方法的时间差异，验证 FFT 的高效性。

实验代码

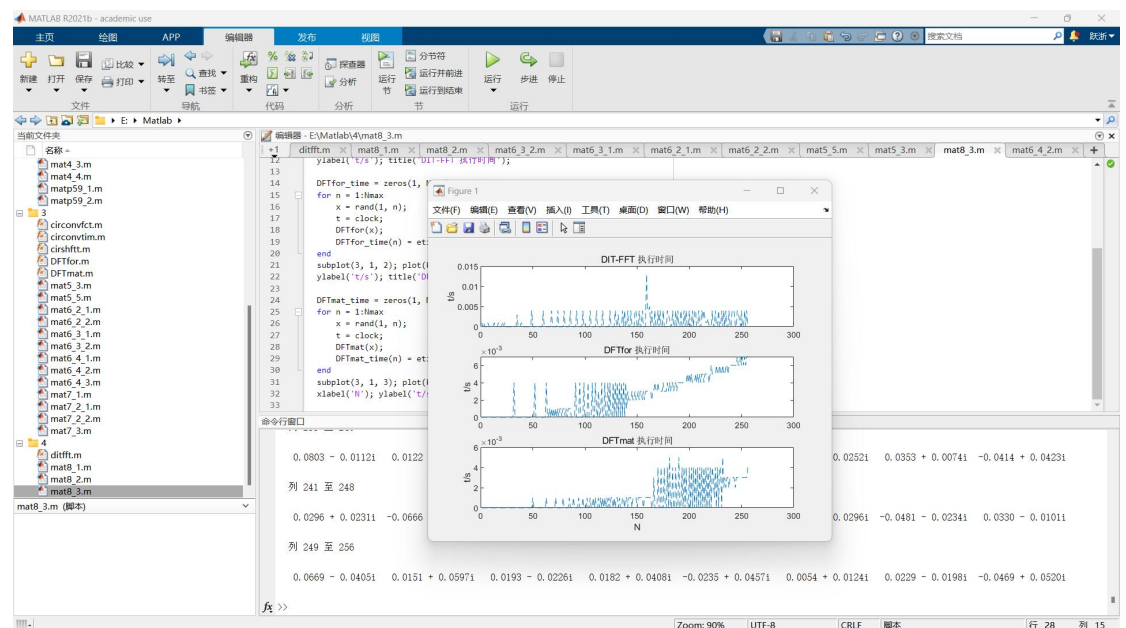
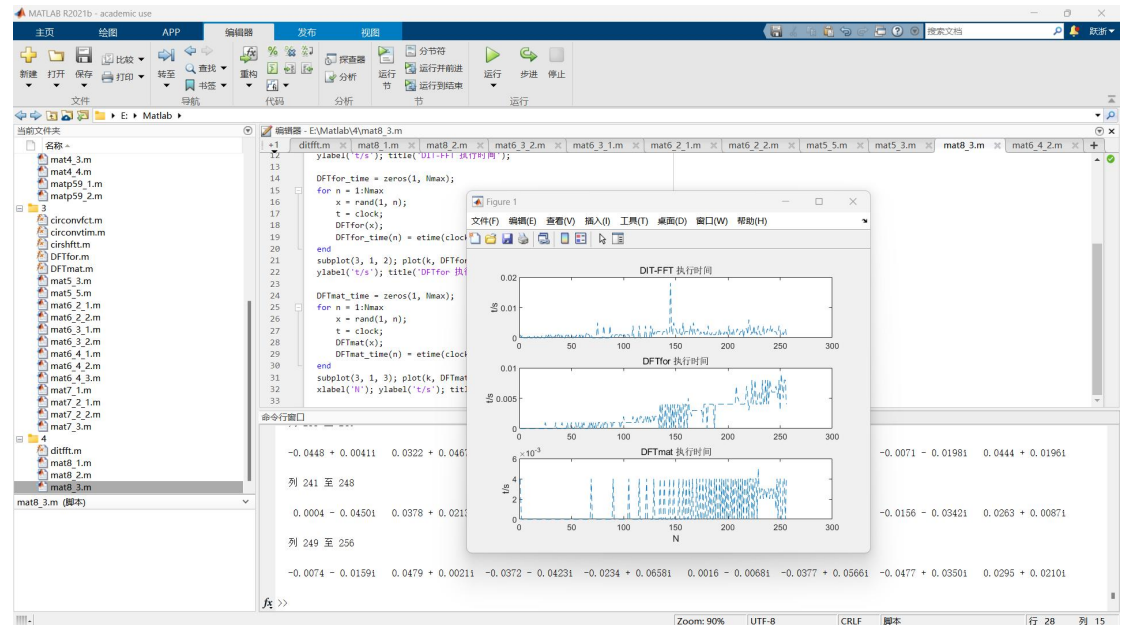
```
clc; clear; close all;
Nmax = 256;
ditfft_time = zeros(1, Nmax);
for n = 1:Nmax
    x = rand(1, n);
    t = clock;
    ditfft(x);
    ditfft_time(n) = etime(clock, t);
end
k = 1:Nmax;
subplot(3, 1, 1); plot(k, ditfft_time, 'b-');
ylabel('t/s'); title('DIT-FFT 执行时间');

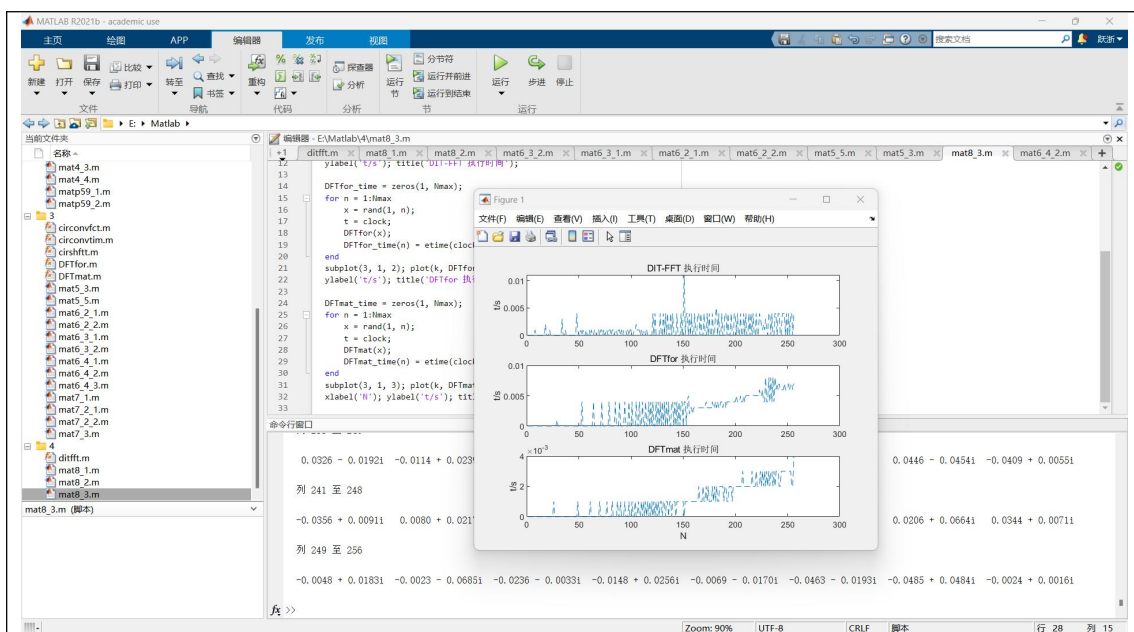
DFTfor_time = zeros(1, Nmax);
for n = 1:Nmax
    x = rand(1, n);
    t = clock;
    DFTfor(x);
    DFTfor_time(n) = etime(clock, t);
end
subplot(3, 1, 2); plot(k, DFTfor_time, 'b-');
ylabel('t/s'); title('DFTfor 执行时间');

DFTmat_time = zeros(1, Nmax);
for n = 1:Nmax
    x = rand(1, n);
    t = clock;
    DFTmat(x);
    DFTmat_time(n) = etime(clock, t);
end
```

```
subplot(3, 1, 3); plot(k, DFTmat_time, '-');
xlabel('N'); ylabel('t/s'); title('DFTmat 执行时间');
```

实验结果





9-1

实验目的

利用快速卷积法计算给定序列 $x(n)$ 和 $h(n)$ 的卷积和 $y(n)$, 并通过绘图展示输入序列和卷积结果。通过实验验证快速卷积法在计算序列卷积中的有效性和准确性, 加深对快速卷积法原理的理解。

实验原理

卷积是信号处理中的一种基本运算, 用于计算两个序列之间的关系。直接计算卷积需要较高的计算复杂度, 而快速卷积法利用快速傅里叶变换 (FFT) 和逆快速傅里叶变换 (IFFT) 将卷积计算转化为频域乘法, 再通过 IFFT 返回时域, 显著提高了计算效率。

具体来说, 给定两个序列 $x(n)$ 和 $h(n)$, 首先对它们进行零填充, 使其长度扩展到两者长度之和减一。然后对零填充后的序列分别进行 FFT 变换, 将其转换到频域。频域中的卷积运算变为对应频率点的乘法, 计算结果再通过 IFFT 变换回到时域, 得到最终的卷积结果 $y(n)$ 。这种方法将时间复杂度从直接计算的 $O(N^2)$ 降低到 $O(N \log N)$, 大大提高了计算效率。通过对比输入序列和卷积结果, 可以验证快速卷积法的有效性。

实验代码

函数 fftconv ()

```
function y=fftconv(x1,x2,N)
```

```
Xk1=fft(x1,N);
```

```
Xk2=fft(x2,N);
```

```
YK=Xk1.*Xk2;
```

```
y=ifft(YK);
```

```
clear all;clc;close all;
```

```
nx=0:14;xn=sin(0.4*nx);
```

```
nh=0:19;hn=0.9.^nh;
```

```
N1=length(xn);N2=length(hn);
```

```
N=N1+N2-1;
```

```
xn=[xn zeros(1,N-N1)];
```

```
hn=[hn zeros(1,N-N2)];
```

```
yn=fftconv(xn,hn, N);
```

```
nn=0:N-1;
```

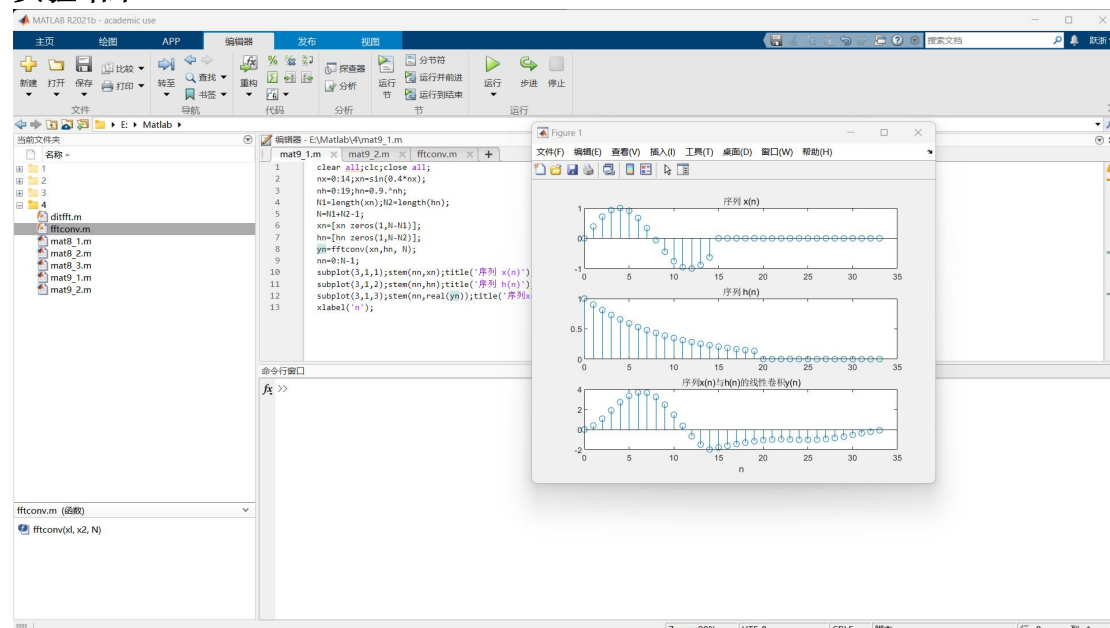
```
subplot(3,1,1);stem(nn,xn);title('序列 x(n)');
```

```
subplot(3,1,2);stem(nn,hn);title('序列 h(n)');
```

```
subplot(3,1,3);stem(nn,real(yn));title('序列 x(n)与 h(n)的线性卷积 y(n)');
```

```
xlabel('n');
```

实验结果



9-2

实验目的

比较直接计算线性卷积 $y(n)=z(n)*x(n)$ 所需的时间和利用快速傅里叶变换 (FFT) 计算 $y(n)=x(n)*x(n)$ 所需的时间, 通过实验验证 FFT 在大规模卷积计算中的高效性。

实验原理

卷积运算是信号处理中的基本操作, 其直接计算的复杂度为 $O(N^2)$, 在处理大规模数据时效率较低。快速傅里叶变换 (FFT) 通过将卷积计算转化为频域中的点乘, 再通过逆快速傅里叶变换 (IFFT) 回到时域, 显著降低了计算复杂度至 $O(N \log N)$ 。通过生成均匀分布的随机数序列和高斯随机序列, 分别采用直接卷积计算和 FFT 方法计算卷积, 并比较两种方法的时间差异, 可以验证 FFT 方法在计算大规模卷积时的高效性。

实验代码

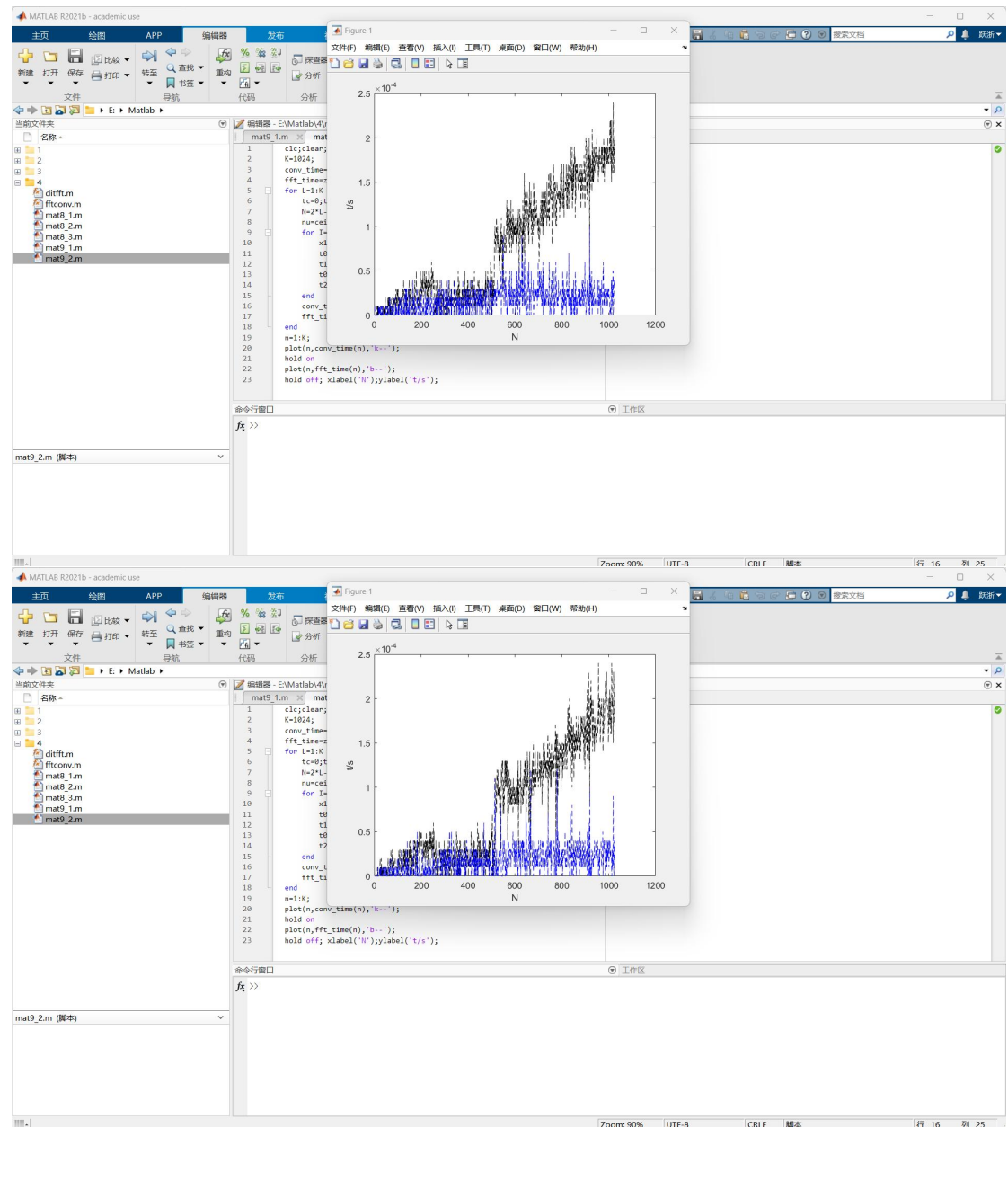
```
clc;clear;close all
K=1024;
conv_time=zeros(1,K);
fft_time=zeros(1,K);
for L=1:K
    tc=0;tf=0;
    N=2*L-1;
    nu=ceil(log10(N)/log10(2));N1=2^nu;
    for I=1:100
        x1=rand(1,L);x2=randn(1,L);
        t0=clock;y1=conv(x1,x2);
        t1=etime(clock,t0);tc=tc+t1;
        t0=clock;Y2=fft(x1,N1).*fft(x2,N1);y2=ifft(Y2,N1);
        t2=etime(clock,t0);tf=tf+t2;
    end
    conv_time(L)=tc/100;
    fft_time(L)=tf/100;
end
```

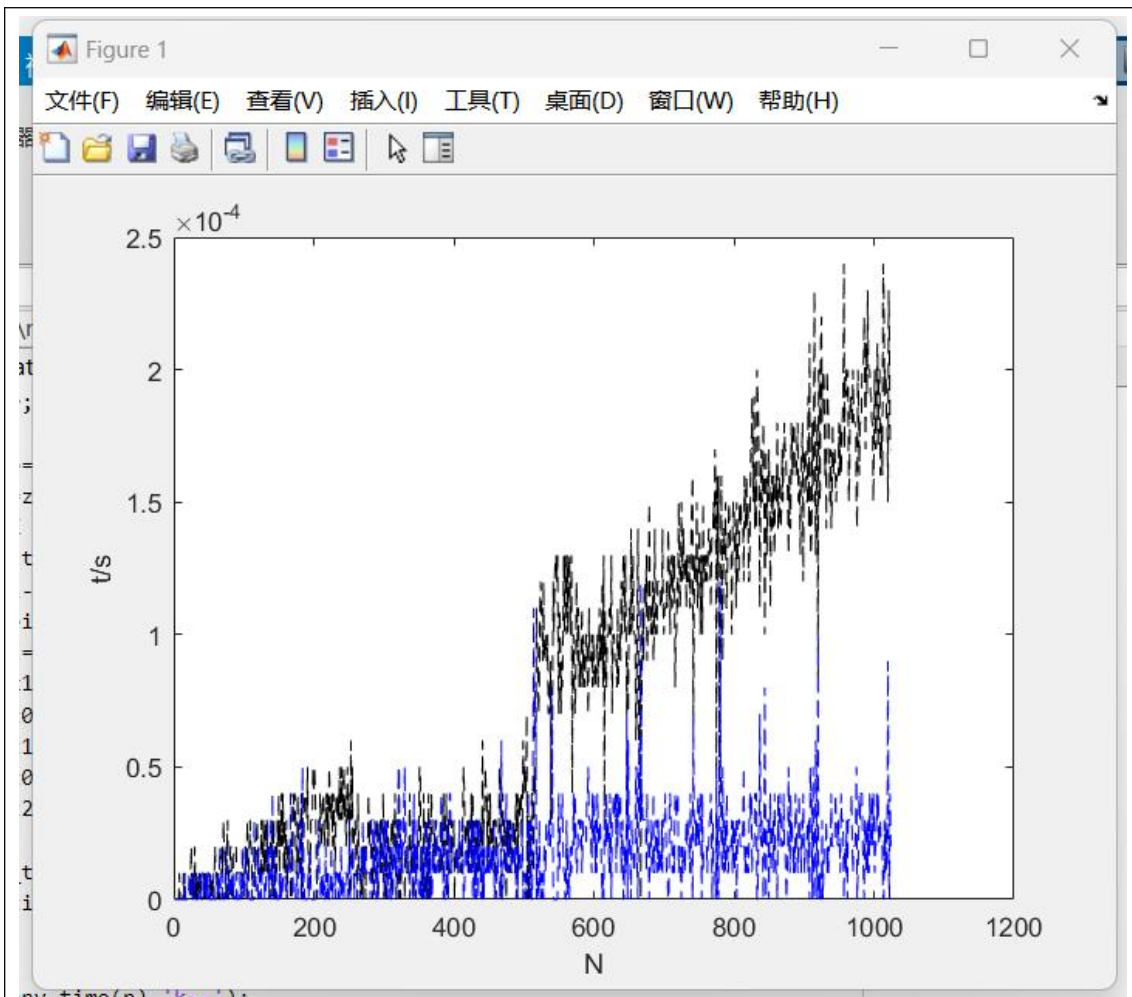
```

n=1:K;
plot(n,conv_time(n),'k--');
hold on
plot(n,fft_time(n),'b--');
hold off; xlabel('N');ylabel('t/s');

```

实验结果





实验总结

通过这次实验，我深刻体会到了快速傅里叶变换（FFT）在信号处理领域的重要性和高效性。在实践中，我不仅掌握了 FFT 及其逆变换（IDFT）的实现方法，还对卷积计算在信号处理中的应用有了更深入的理解。

首先，通过编程实现 FFT，我深入理解了其蝶形运算的原理和过程。手动计算给定序列的频域表示，并与 MATLAB 内置函数进行对比，使我认识到 FFT 的准确性和效率。这一过程不仅强化了我对理论知识的掌握，还提升了动手解决实际问题的能力。

其次，实验中我实现了逆傅里叶变换，并验证了其有效性。这使我进一步了解了傅里叶变换的对称性原理，尤其是如何利用 FFT 来实现高效的逆变换。这对我在实际应用中处理频域与时域信号的转换提供了宝贵的经验。

通过比较直接计算与 FFT 计算的执行时间，我直观地感受到 FFT 在处理大规模数据时的显著优势。这一发现不仅让我认识到算法选择对计算效率的影响，还启发我在今后的研究中，更加注重算法优化和高效计算方法的应用。

在卷积计算实验中，我采用快速卷积法，通过频域乘法实现了高效的卷积运算。这一过程使我深刻理解了卷积运算的基本原理及其在信号处理中的广泛应用。同时，通过验证快速卷积法的准确性，我进一步认识到频域处理方法在提升计算效率方面的巨大潜力。

综合这次实验的各个方面的，我不仅巩固了理论知识，还提升了实践技能，特别是在信号处理和快速算法实现方面的能力。今后，我将把这些经验应用到更多的实际问题中，持续探索和优化高效计算方法，为我的之后的工作提供有力支持。

代码调试过程:

遇到的问题: 发现运行结果和书本上的不完全一样

解决方法: 多次运行，运行时间可能与计算机有关，不完全相同是正常的。实验结果每次不一样且每台电脑也不同, 主要原因包括随机数生成、计算机性能差异、系统负载和背景进程的影响、MATLAB 内置函数的优化以及测量误差。每次运行实验时生成的随机数序列不同会导致输入数据不同, 而不同计算机的硬件性能和系统负载也会影响执行时间。此外，MATLAB 不同版本的函数优化策略和时间测量的精度也会带来一些误差。为了提高结果的稳定性，可以多次运行实验并取平均值。

如题 8-1 的多次运行结果

