

1 习题

1.1 离散傅里叶变换对 (10 分)

若常数项 $\frac{1}{MN}$ 包含在 DFT 里:

$$\text{DFT: } f(x, y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

$$\text{Inverse DFT: } F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

$$\text{则有: } F(0, 0) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)$$

可求原函数的均值 A, 再调用函数接口求得 F(0,0)的值,

若 $A = F(0, 0)$, 则可确认常数项 $\frac{1}{MN}$ 在 DFT 中。

同理:

若 $A = \frac{1}{MN} F(0, 0)$, 常数项 $\frac{1}{MN}$ 在 IDFT 中。

若 $A = \frac{1}{\sqrt{MN}} F(0, 0)$, 常数项 $\frac{1}{\sqrt{MN}}$ 分别在正变换和反变换前面。

1.2 傅里叶频谱 (15 分)

Ans: 对应的傅里叶频谱是一样的

理由:

直接代入式(4.5-15)和式(4.5-16)可以证明傅里叶变换对满足下列平移特性(见习题 4.16):

$$f(x, y) e^{j2\pi(u_0 x/M + v_0 y/N)} \Leftrightarrow F(u - u_0, v - v_0) \quad (4.6-3)$$

和

$$f(x - x_0, y - y_0) \Leftrightarrow F(u, v) e^{-j2\pi(x_0 u/M + y_0 v/N)} \quad (4.6-4)$$

也就是说, 用指数项乘以 $f(x, y)$ 将使 DFT 的原点移到点 (u_0, v_0) ; 反之, 用负指数乘以 $F(u, v)$ 将使 $f(x, y)$ 的原点移到点 (x_0, y_0) 。正如我们在例 4.13 中说明的那样, 平移不影响 $F(u, v)$ 的幅度(谱)。

由于指数项的绝对值是 1, 谱对图像平移是不敏感的。

由题, 对(a)进行延拓, 填充相同数量 0, 仅位置不同, 即(c)图相当于对(b)图进行平移, 可知对应的傅里叶频谱必然相同。

1.3 频率域滤波器

1. (5 分) 找出一个滤波器 $H(u, v)$ ，使得它在频率域与以下空间滤波器等价

$$\frac{1}{4} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 1.0000 + 0.0000i & -0.1250 - 0.2165i & -0.1250 + 0.2165i \\ -0.1250 - 0.2165i & 0.2500 - 0.4330i & -0.5000 + 0.0000i \\ -0.1250 + 0.2165i & -0.5000 + 0.0000i & 0.2500 + 0.4330i \end{bmatrix}$$

2. (10 分) $H(u, v)$ 是高通滤波器还是低通滤波器？

Ans:

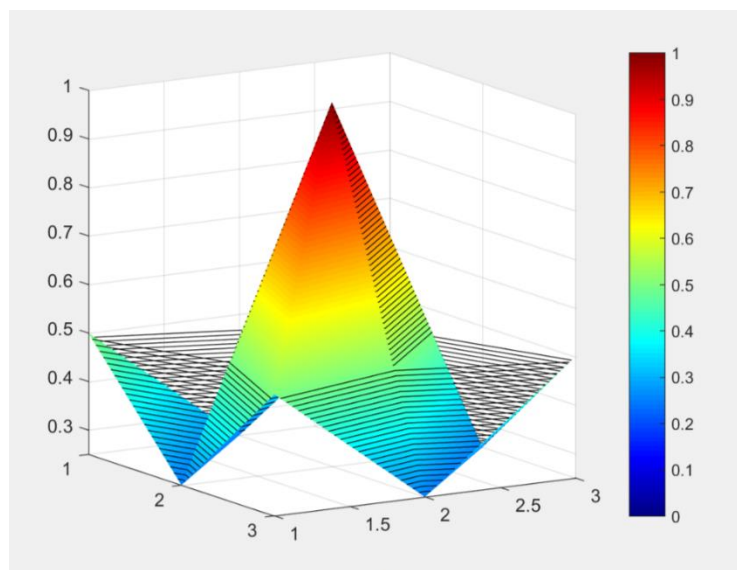
是低通滤波器。

证明：对得到的 $H(u, v)$ 取模，则 $H(0, 0)$ 处相当于原矩阵乘以 $(-1)^{x+y}$ 再傅里叶变换使频率域滤波器中心化后的滤波器中心。

取模后结果： $\begin{bmatrix} 1 & 0.25 & 0.25 \\ 0.25 & 0.5 & 0.5 \\ 0.25 & 0.5 & 0.5 \end{bmatrix}$ 可知 $H(0, 0)$ 处是傅里叶变换系数频谱矩阵最大值，即

通过低频，衰减高频，可知此滤波器为低通滤波器。

用 matlab 对中心化后的频谱绘制三维图像：



中心化后的傅里叶频谱矩阵

filter =

0.5000	0.2500	0.5000
0.2500	1.0000	0.2500
0.5000	0.2500	0.5000

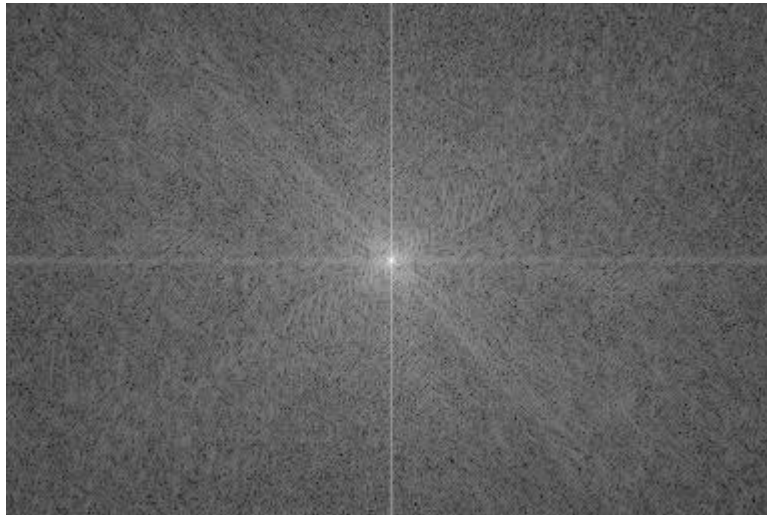
(程序见 exercise.m)

2. 编程题

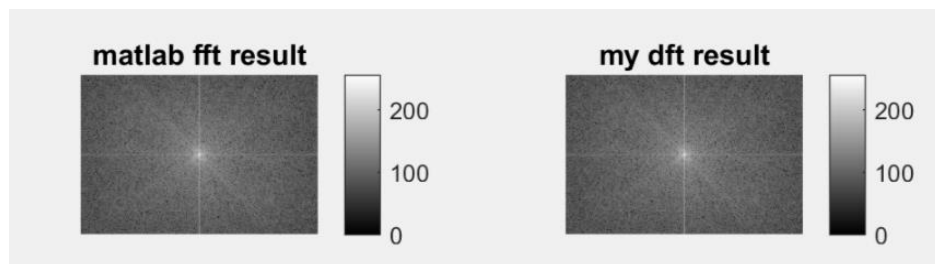
2.2 傅里叶变换 (30 分)

1. (10 分) 对输入图像做离散傅里叶变换 (DFT)，并且把中心化后的傅里叶频谱粘贴到报告里。

中心化后的傅里叶频谱图：



运行效果对比图：

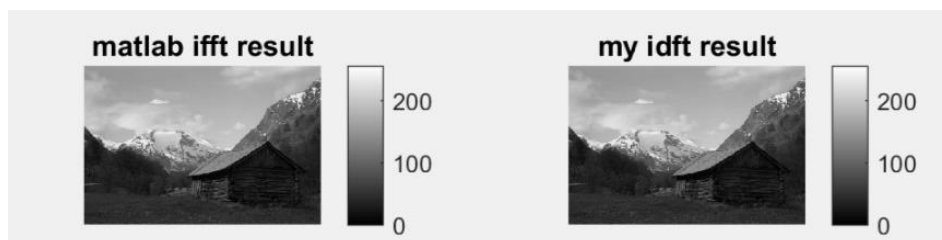


2. (5 分) 对上一个问题的结果做离散傅里叶反变换 (IDFT)，并把实部粘贴到报告里。 注意：实部与输入图像非常相似。(请思考其中的原因。)

反变换得到的实部图像：



(选择实部忽略了由于计算不准确导致的寄生复分量，与输入图像十分相似)
运行效果对比图：



3. (15 分) 详细描述你是如何实现 DFT / IDFT 的，也就是说，针对“dft2d”函数进行算法说明，字数不能超过两页。请集中在算法方面，不要简单地复制/粘贴代码。

(1) 根据二维傅里叶函数的可分性

The 2-D DFT can be implemented as TWO 1-D DFT as follows

$$\begin{aligned}
 F(u, v) &= \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j\left(\frac{2\pi ux}{M} + \frac{2\pi vy}{N}\right)} \\
 &= \frac{1}{M} \sum_{x=0}^{M-1} f(x, y) e^{-j\left(\frac{2\pi ux}{M}\right)} \cdot \sum_{y=0}^{N-1} f(x, y) e^{-j\left(\frac{2\pi vy}{N}\right)} \\
 &= \frac{1}{M} \sum_{x=0}^{M-1} F(x, v) e^{-j\left(\frac{2\pi ux}{M}\right)}
 \end{aligned}$$

Therefore the 2-D DFT can be separated into two 1-D DFT as follows

(1) Obtain $F(x, v) = \sum_{y=0}^{N-1} f(x, y) e^{-j\left(\frac{2\pi vy}{N}\right)}$ using a 1-D DFT along y-axis
(i.e., row operation)

(2) Obtain $F(u, v) = \frac{1}{M} \sum_{x=0}^{M-1} F(x, v) e^{-j\left(\frac{2\pi ux}{M}\right)}$ using a 1-D DFT along x-axis
(i.e., column operation)

(2) 利用傅里叶变换的共轭性质，由正变换公式计算反变换

主要思路是基于傅里叶变换的共轭性质，利用正变换公式计算反变换。
傅里叶变换和反变换

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) e^{-j2\pi ux/M}$$

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{j2\pi ux/M}$$

上式两边取复共轭并除以M, 有

$$\frac{1}{M} f^*(x) = \frac{1}{M} \sum_{u=0}^{M-1} F^*(u) e^{-j2\pi ux/M}$$

上式的形式和前向变换一样，左边再取复共轭并乘以M就是对F(u)反变换的函数。

(3) 一维傅里叶变换的实现(dft1d.m)

由(1)(2)可知，最重要的部分是实现一维傅里叶变换，而后只需逐行变换再用变换结果逐列变化即可得到傅里叶变换后的系数矩阵。

$$1d\text{-DFT} : F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) e^{-j2\pi ux/M}, u = 0, 1, \dots, M-1$$

对外层循环 0 到 M-1 范围内的每个 u，都有内层循环 x（范围 0 到 M-1）使满足上述求和公式得到对应结果。

例：u = 0 时：

$$\begin{bmatrix} F(0) \\ F(1) \\ \vdots \\ F(M-2) \\ F(M-1) \end{bmatrix} = \frac{1}{M} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_M^1 & W_M^2 & \dots & W_M^{M-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & W_M^{M-2} & W_M^{2(M-2)} & \dots & W_M^{(M-1)(M-2)} \\ 1 & W_M^{M-1} & W_M^{2(M-1)} & \dots & W_M^{(M-1)(M-1)} \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(M-2) \\ f(M-1) \end{bmatrix}$$

```
dft1d.m
1 function output_vector = dft1d( input_vector)
2     M = length(input_vector);
3     output_vector = zeros(1,M);
4     input_vector = double(input_vector);
5     for u = 0:M-1
6         for x = 0:M-1
7             f_x = input_vector(x+1);
8             item = (-1i)*2*pi*u*x/M;
9             Wn = exp(item);
10            output_vector(u+1) = output_vector(u+1) + f_x * Wn;
```

(4) 由一维傅里叶变换扩展到二维(dft2d.m)

- 1) 遍历复共轭矩阵的每一行，调用一维傅里叶变换函数，用变换结果代替原行。
- 2) 遍历矩阵每一列，调用一维傅里叶变换函数，用变换结果代替原列

```
for v=1:M                                for u=1:N
    % row operation                        % col operation
    row_item = input_img(v,:);            col_item = output_img(:,u);
    dft_row = dft1d(row_item);            dft_col = dft1d(col_item);
    output_img(v,:) = dft_row;             output_img(:,u) = dft_col;
end                                        end
```

(5) 由二维傅里叶正变换得到反变换(dft2d.m)

- 1) 获得傅里叶变换所得结果的复共轭矩阵

```
dft1d.m dft2d.m
1 function output_img = dft2d( input_img, flag )
2     if flag == 1
3         input_img = conj(input_img);
4     end
```

- 2) 对此矩阵做正变换
- 3) 对正变换结果再取复共轭，得到反变换结果

4) 对最终结果乘常数项 $1/MN$ (本程序中常数项在反变换中)

```
if flag == 1
    output_img = output_img / (M*N);
    output_img = conj(output_img);
end
```

注：

dft2d.m 对应函数格式为 `function output_img = dft2d(input_img,flag)`

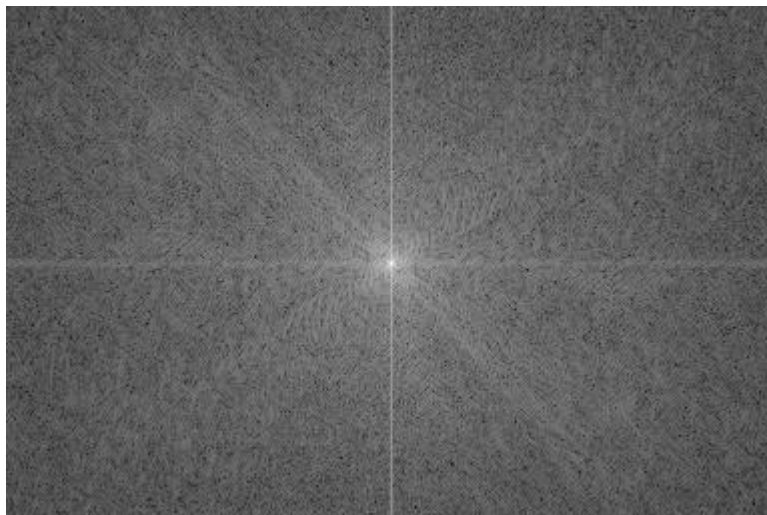
当 `flag = 0` 时，做傅里叶正变换

当 `flag = 1` 时，做傅里叶反变换

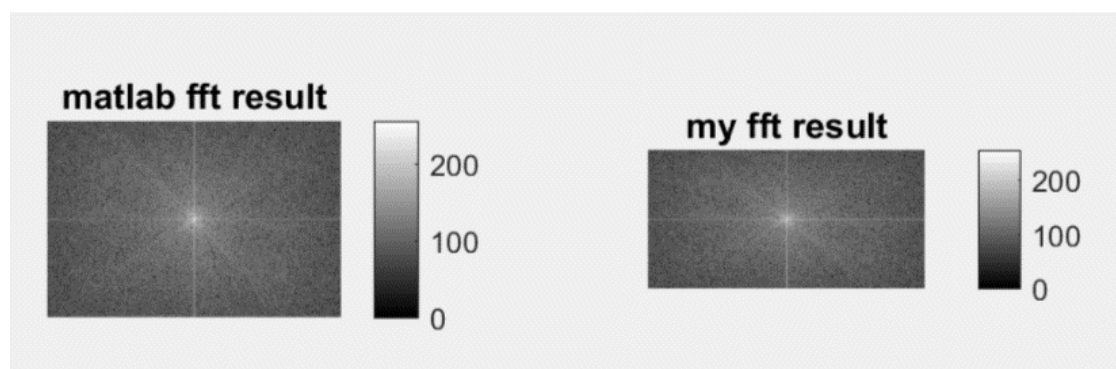
2.3 附加题：快速傅里叶变换 (50 分)

1. (15 分) 对输入图像做快速傅里叶变换 (FFT)，并且把中心化后的傅里叶频谱粘贴到报告里。

中心化后的傅里叶频谱图：



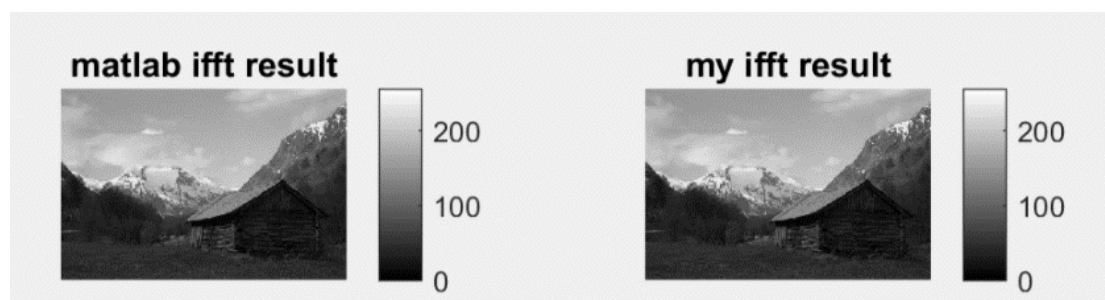
运行效果对比图：



2. (10 分) 对上一个问题的结果做快速傅里叶反变换 (IFFT)，并把实部粘贴到报告里。
快速傅里叶反变换结果：



运行效果对比图：



3. (25 分) 详细描述你是如何实现 FFT / IFFT 的，也就是说，针对“fft2d”函数进行算法说明，字数不能超过三页。

算法描述

与 DFT 同理，二维 DFT 可以使用逐次通过一维变换的方法来执行，重点考虑如何实现一维 FFT。

$$F(u) = \sum_{x=0}^{M-1} f(x) W_M^{ux}, u = 0, 1, \dots, M-1$$

其中 $W_M = e^{-j2\pi/M}$ ，假设 $M=2^n$ ，可表示为 $M=2K$

$$\text{由旋转因子的性质可知：} F(u) = \sum_{x=0}^{K-1} f(2x) W_K^{ux} + \sum_{x=0}^{K-1} f(2x+1) W_K^{ux} W_{2K}^{ux}$$

$$\text{定义 } F_{\text{even}}(u) = \sum_{x=0}^{K-1} f(2x) W_K^{ux}, F_{\text{odd}}(u) = \sum_{x=0}^{K-1} f(2x+1) W_K^{ux}, u = 0, 1, 2, \dots, K-1$$

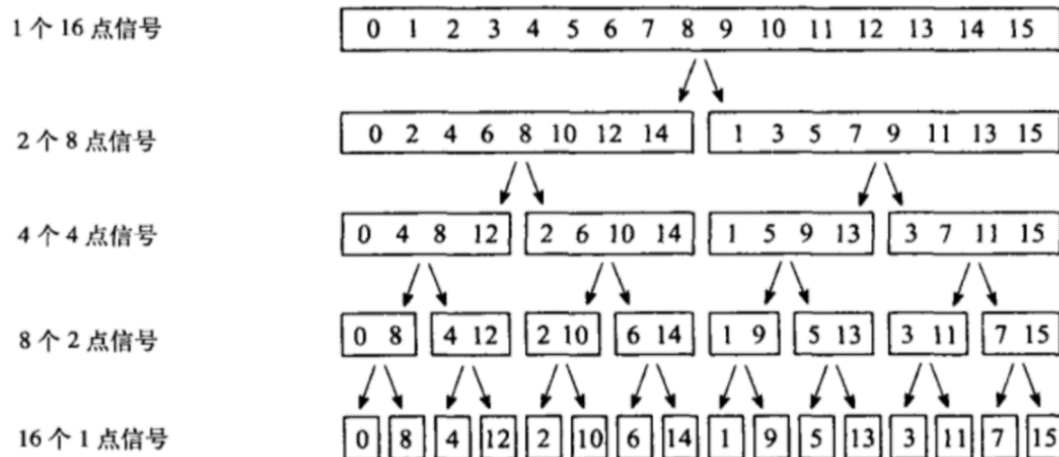
$$\text{可得：} F(u) = F_{\text{even}}(u) + F_{\text{odd}}(u) W_M^u, F(u+K) = F_{\text{even}}(u) - F_{\text{odd}}(u) W_M^u$$

(1) $n = \log_2 M$ 步分解

以 $M=16$ 的 16 点信号为例：

一个 16 点信号要经过 4 次分解，先分为 2 个 8 点信号，再分为 4 个 4 点信号，如此进行下去，直到得到 $M=16$ 个由 1 点组成的信号。

每次信号一分为二时使用交错分解，即把信号分为偶数抽样点和奇数抽样点。

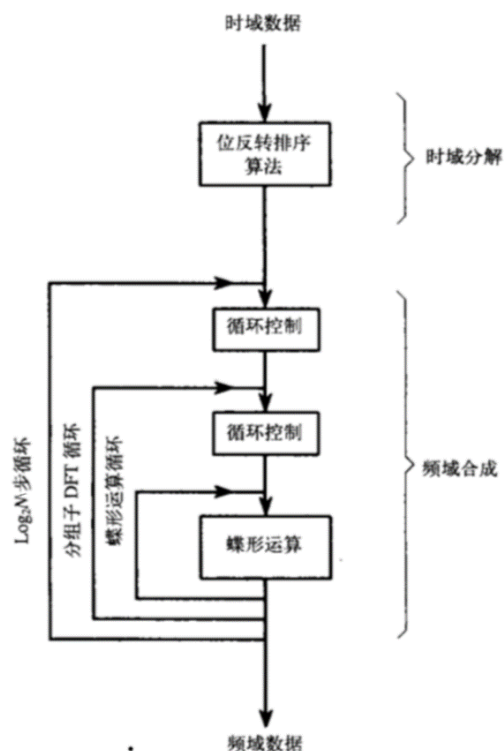


(2) 二进制位反转

由分解的结构可以知道此分解实际上是对信号抽样点的重新排列，左边和右边分别为原始信号抽样点和重新排列的信号点及其对应的二进制。

FFT 的时域分解可以通过二进制反转算法（把 M 个时域信号抽样点序号的二进制数值从左到右反转）实现。

正常顺序的样点序号		位反转后的样点序号	
十进制	二进制	十进制	二进制
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0100
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011
14	1110	7	0111
15	1111	15	1111



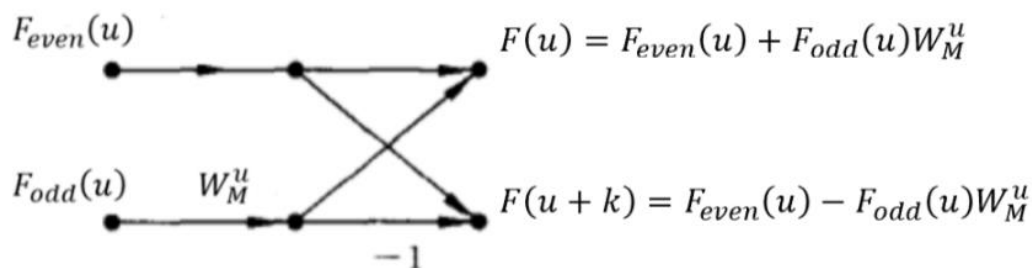
(3) 找到 1 点时域信号的频谱

最底层 1 点信号的频谱等于它本身，将此时域数据转换到频域不需进行任何操作。（此刻其本身可看作频谱）

(4) 从底层逐阶向上合并

FFT 流程分三步：①把一个 M 点时域信号分解为 M 个单点信号②计算这 M 个单点信号频谱（等于它本身）③把这 M 个频谱合成一个频谱

1) 两点蝶形算法（M=2,K=1）



2) 合成频域信号获得傅里叶变换的最终结果，需要三层循环。

① 最外层循环：共有 $\log_2(M)$ 次向上的处理，总阶数 $n = \log_2(M)$

② 中间层循环：每一次向上合成得到的分组个数，u 阶组数： $2^{(n-u)}$

③ 最内层循环：合成每个分组需要的蝶形计算次数，u 阶每组蝶形次数： $2^{(u-1)}$

a) 内层循环单元

最内层循环内是一个两点蝶形运算单元。

```

19 % 最外层循环：共有 log2(M) 次向上的处理
20 for u = 1 : n
21     pos = 0;
22     % 中间层循环：每一次向上合成得到的分组个数
23     for g = 1 : 2^(n - u)
24         % 最内层循环：合成每个分组需要的蝶形计算次数
25         for b = 1 : 2^(u-1)
26             W = exp((-1j)*(2*pi*(b-1)*2^(n - u)/M));

```

b) 蝶形运算的旋转因子 W_M^u 确定方法：

每次蝶形运算，奇数项的旋转因子 W_M^u 都会更新。

合成每个分组做 b 次蝶形运算，则每次循环时指数 u 随组内蝶形运算进行次数从 0 到 b-1 依次递增，可知旋转因子指数系数 $u = b-1$

(注意此处的 u 并非外层循环阶数)

每次降阶，M 的值都要减半，相当于在 u 阶时， $M = M/2^{(n-u)}$

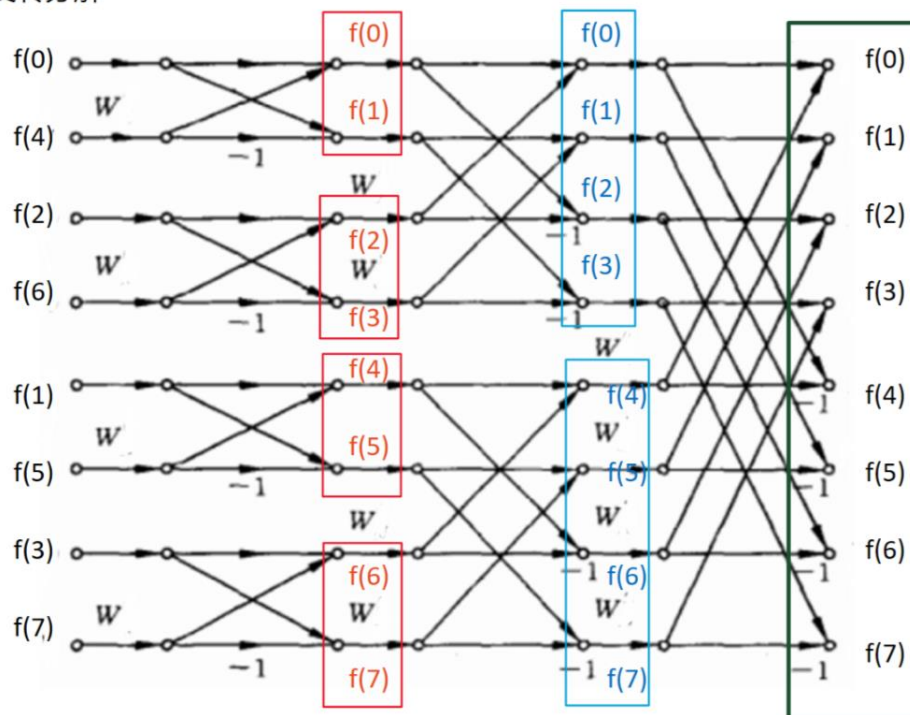
$$\text{可知： } W_M^u = e^{-j2\pi u/M} = e^{\frac{-j2\pi(b-1)}{M/2^{(n-u)}}} = e^{-j2\pi(b-1)2^{(n-u)}/M}$$

c) 程序使用置换计算，同一个数组 f 用于记录各阶中间结果及最终频域结果的值。

b) 联系实现方式进行具体描述：8 点 DFT 合成：

$M=8$
 $n=3$
 反转分解

1阶向上 $u=1$	2阶向上 $u=2$	3阶向上 $u=3$
合成得到分组个数 $g=2^{n-u}=4$	合成得到分组个数 $g=2^{n-u}=2$	合成得到分组个数 $g=2^{n-u}=1$
合成每个分组所需蝶形	合成每个分组所需蝶形	合成每个分组所需蝶形
计算次数 $b=2^{u-1}=1$	计算次数 $b=2^{u-1}=2$	计算次数 $b=2^{u-1}=4$



2.4 频率域滤波 (30 分)

1. (12 分) 分别用 3×3 , 7×7 和 11×11 的均值滤波器来平滑你输入的图像, 将相应的三个输出结果 粘贴到报告里。

(1) 3×3 均值滤波



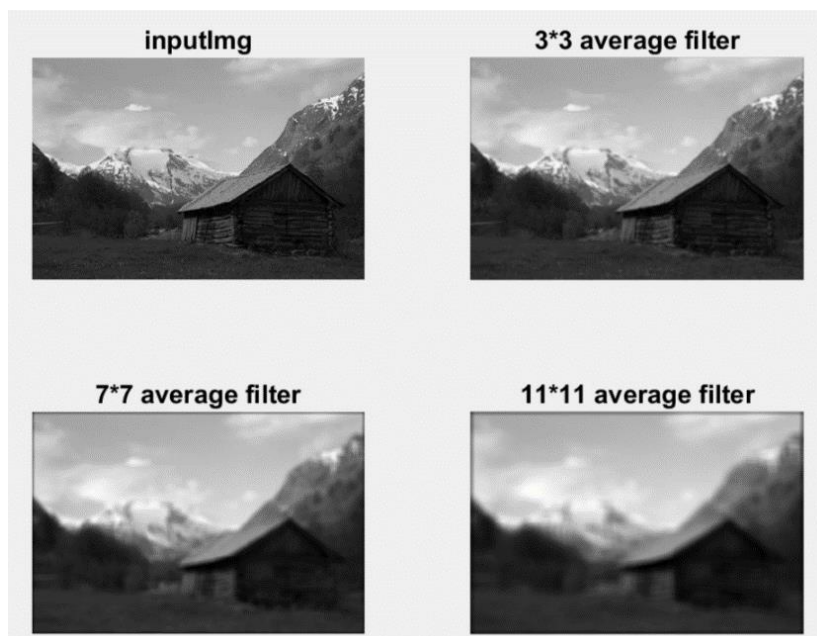
(2) 7 x7 均值滤波



(3) 11 x 11 均值滤波



(4) 运行效果对比图



2. (4 分) 用 3×3 的拉普拉斯滤波器来锐化你输入的图像 (本上有 4 种拉普拉斯滤波器, 参见图 3.37, 你可以使用其中任意一种), 并将输出结果放在报告中。

(1) 选择的拉普拉斯滤波器

0	-1	0
-1	4	-1
0	-1	0

(2) 拉普拉斯滤波器处理结果



(3) 拉普拉斯结果加原图得到锐化后图像



(4) matlab 程序运行结果对比图



3. (14 分) 详细描述你是如何实现滤波操作的, 也就是说, 针对“filter2d_freq”函数进行算法说明, 字数不能超过两页。

(1) 给定一幅图像($M \times N$)和一个滤波器矩阵($A \times B$), 得到填充参数 p, q 。

若对 $f(x, y)$ 和 $h(x, y)$ 分别是大小为 $A \times B$ 和 $C \times D$ 的矩阵, 缠绕错误可以通过零填充来避免, 方法如下:

$$f_p(x, y) = \begin{cases} f(x, y), & 0 \leq x \leq A-1 \text{ 和 } 0 \leq y \leq B-1 \\ 0, & A \leq x \leq P \text{ 或 } B \leq y \leq Q \end{cases}$$

和

$$h_p(x, y) = \begin{cases} h(x, y), & 0 \leq x \leq C-1 \text{ 和 } 0 \leq y \leq D-1 \\ 0, & C \leq x \leq P \text{ 或 } D \leq y \leq Q \end{cases}$$

其中,

$$P \geq A + C - 1$$

和

$$Q \geq B + D - 1$$

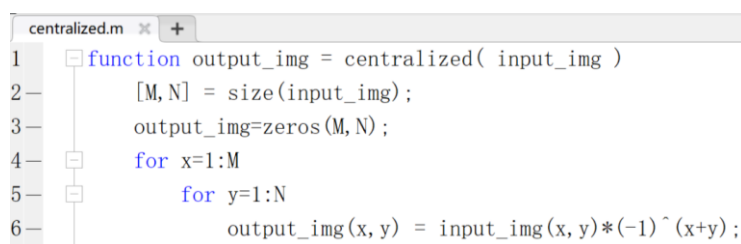
根据以上规律, 取 $P = M + A - 1, Q = N + B - 1$

(2) 对 $f(x, y)$ 添加必要数量的 0, 形成大小为 $P \times Q$ 的填充后的图像 $f_p(x, y)$

```
padding_f = zeros(P, Q);
padding_f(1:M, 1:N) = input_img;
%中心化
padding_f = centralized(padding_f);
```

(3) 用 $(-1)^{x+y}$ 乘以 $f_p(x, y)$ 移到其变换的中心

```
%中心化
padding_f = centralized(padding_f);
```



```
centralized.m  x +
1 function output_img = centralized( input_img )
2     [M,N] = size(input_img);
3     output_img=zeros(M,N);
4     for x=1:M
5         for y=1:N
6             output_img(x,y) = input_img(x,y)*(-1)^(x+y);
```

(4) 计算来自步骤(3)的图像的 DFT, 得到 $F(u, v)$

```
%计算输入图像中心化后的DFT -> F(u, v)
% F = fft2(padding_f);
F = fft2d(padding_f, 0);
% F = dft2d(padding_f, 0);
```

(5) 生成一个实的，对称的滤波函数 $H(u, v)$ ，其大小为 $P \times Q$ ，中心在 $(P/2, Q/2)$ 处。(此过程与 $F(u, v)$ 生成过程一致。用阵列相乘形成乘积 $G(u, v) = H(u, v)F(u, v)$ ；即 $G(i, k) = H(i, k)F(i, k)$ 。

$H(u, v)$ 生成与上述过程同理。

```
%用阵列相乘形成乘积G
G = F.* H;
%傅里叶逆变换求出处理后的图像
%    g_p = ifft2(G);
    g_p = fft2d(G, 1);
%    g_p = dft2d(G, 1);
```

(6) 得到处理后的图像：

$$g_p(x, y) = \left\{ \text{real} \left[\mathcal{F}^{-1} [G(u, v)] \right] \right\} (-1)^{x+y}$$

其中，为忽略由于计算不准确导致的寄生复分量，选择了实部，下标 p 指出我们处理的是填充后的阵列。

```
%为忽略由于计算不准确导致的寄生复分量，选择了实部
g_p = real(g_p);
%此时得到填充后的阵列g_p(x, y)
g_p = centralized(g_p);
```

(7) 通过从 $g_p(x, y)$ 的左上象限提取 $M \times N$ 区域，得到最终处理结果 $g(x, y)$ 。

(8) 滤波结果的暗边优化处理

滤波后图像的暗边是使用零填充处理后由低通滤波造成的。

如果不经处理，输出图像的左侧和上方会产生黑边，且随着空间均值滤波器尺寸的增大，黑边会更加明显。

可以在第(7)步中，将左上象限 $M \times N$ 区域提取的起始位置向右下移动，使起始点由 $(1, 1)$ 变为 $(A/2 + 1, B/2 + 1)$ 。

```
% 从g(x, y)的左上象限提取M*N区域，得到最终处理结果g(x, y)
% 对不同尺寸的滤波器要最终结果要进行相应平移，不能直接从(1, 1)开始取
start_posx = floor(A/2)+1;
start_posy = floor(B/2)+1;
output_img = g_p(start_posx:start_posx+M-1, start_posy:start_posy+N-1);
```