数字图像处理
第二次小作业报告

自96 曲世远 2019011455

Problem1

1.算法要点与理论原理

我认为本次作业第一题主要有以下几个要点：

1. 首先需要生成满足题目要求的二位波形图形，并且可以依据参数进行调节；
2. 针对题目要求的五种波形需要设计出对应于波形的参数调整UI界面；
3. 实现DFT运算函数并且保证运算速度；
4. 实现较为美观且准确的结果显示

针对以上我认为以及实现过程中遇到的算法要点，我才用了以下的算法原理加以解决：

1. 将波形产生函数进行整合，并基于$Wavetype$变量进行波形发生；
2. 由于如果采用但界面+隐藏UI设计会导致很大的重复代码量且不利于调整界面设计以及不够直观；因此我使用了$Matlab\ app\ designer$中的选项卡组容器，容纳不同波形选项时的参数调整组件。
3. 实现了基于矩阵的DFT运算函数以提高运算速度并且保证正确性，同时我引入了系统内置的FFT函数可以进行两种变换的比较，以验证正确性。
4. 由于surf并不能直接显示在$Matlab\ app\ designer$上，因此需要通过坐标图，并将surf加载到坐标图上。但由于$Matlab\ app\ designer$中的坐标图组件参数过少，并不能指定坐标轴的刻度显示精度，因此我设计了精确显示模式（可能会因为小数点后位数过多导致显示不美观）与非精确显示模式。

2.算法具体实现

```
%% generate origin figure
function Wave_generate(app)
    fig = zeros(256, 256);
    fig_1 = zeros(256, 256); %#ok<PREALL>
    fig_2 = zeros(256, 256); %#ok<PREALL>
    [X, Y] = meshgrid(1: 256);
    switch app.Wavetype
    case "Delta"
        fig(257 - app.position_y, app.position_x) = 1;
    case "Sine"
        an = app.angle_sin .* pi ./ 180;
        ph = app.phase .* pi ./ 180;
        fig = cos(2 .* pi .* app.frequency .* (cos(an) .* X + sin(an) .* Y)
+ ph);
    case "Rectangle"
        % have to edit the logic of rotate
        y_min = ceil(app.center_x - app.length ./ 2);
        y_max = ceil(app.center_x + app.length ./ 2);
        x_min = ceil(app.center_y - app.width ./ 2);
        x_max = ceil(app.center_y + app.width ./ 2);
        x_min = app.mapping1_256(x_min);
        x_max = app.mapping1_256(x_max);
        y_min = app.mapping1_256(y_min);
        y_max = app.mapping1_256(y_max);
```

```
        fig(x_min : x_max, y_min: y_max) = 1;
        fig = imrotate(fig, app.angle_rec, "bilinear", "crop");
    case "Gauss"
        fig = (X - 128.5) .^ 2 + (Y - 128.5) .^ 2;
        fig = exp(- fig ./ (2 .* app.variance)) ./ (2 .* pi .*
app.variance);
    case "Gabor"
        an = app.angle_gabor .* pi ./ 180;
        ph = app.phase_gabor .* pi ./ 180;
        fig_1 = cos(2 .* pi .* app.frequency_gabor .* (cos(an) .* X +
sin(an) .* Y) + ph);
        fig_2 = (X - 128.5) .^ 2 + (Y - 128.5) .^ 2;
        fig_2 = exp(- fig_2 ./ (2 .* app.variance_gabor)) ./ (2 .* pi .*
app.variance_gabor);
        fig = fig_1 .* fig_2;
    end
    app.Origin_fig_2 = fig;
end
```

首先通过不同波形的生成函数生成二维波形图像$app.Origin_fig_2$。

```
%% generate the DFT figure
function Ffigure = DFT_2(~, figure)
    [M, N] = size(figure);
    ux = (0 : M - 1)' * (0 : M - 1);
    vy = (0 : N - 1)' * (0 : N - 1);
    eMUX = exp(-2 * pi * 1i / M) .^ ux;
    eNVY = exp(-2 * pi * 1i / N) .^ vy;
    figure = figure + 0i;
    Ffigure = eMUX * figure * eNVY;
end
```

本段代码是进行DFT变换的核心代码，实现了一个对于二维$figure$图像的DFT变换。具体实现逻辑为先将$e^{-j2\pi(ux/M+vy/N)}$转化为$eMUX * eNVY$，而$eMUX, eNVY$的生成过程也是通过向量运算与矩阵运算实现的，因此整个运算过程速度可以得到保证。之后利用$eMUX * figure * eNVY$的公式就可以计算得到$figure$的DFT变换结果。

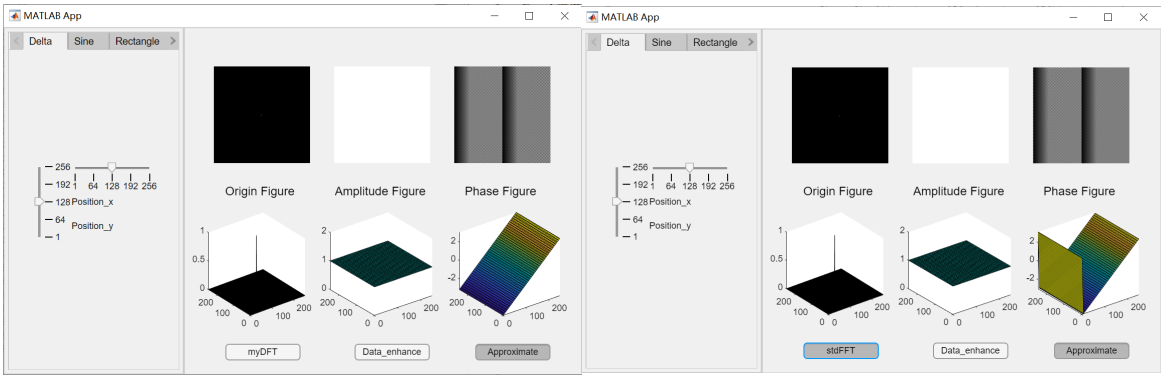
```
%% show the 3D surf
function show_3D(app)
    step = 6;% to present better
    [X, Y] = meshgrid(1 : 256, 1 : 256);
    [X_step, Y_step] = meshgrid(1: step: 256, 1 : step : 256);
    switch app.Wavetype
    case "Delta"
        surf(app.Axes_origin, X, Y, app.Origin_fig_2);
        surf(app.Axes_ampli, X_step, Y_step, app.Amplitude_fig_2(1 : step :
256, 1 : step : 256));
        surf(app.Axes_phase, X_step, Y_step, app.Phase_fig_2(1 : step : 256,
1 : step : 256));
    case {"Sine", "Rectangle", "Gauss", "Gabor"}
        surf(app.Axes_origin, X_step, Y_step, ...
        app.Origin_fig_2(1 : step : 256, 1 : step : 256));
        surf(app.Axes_ampli, X_step, Y_step, ...
        app.Amplitude_fig_2(1 : step : 256, 1 : step : 256));
        surf(app.Axes_phase, X_step, Y_step, ...
        app.Phase_fig_2(1 : step : 256, 1 : step : 256));
```

```
    end
end
%function to show the 6 figure
function fig_show(app)
    app.Wave_generate();
    if app.myDFTButton.Text == "myDFT"
        DFT_origin = app.DFT_shift(app.DFT_2(app.Origin_fig_2));
    elseif app.myDFTButton.Text == "stdFFT"
        DFT_origin = fftshift(fft2(double(app.Origin_fig_2)));
    end
    app.Amplitude_fig_2 = abs(DFT_origin);
    if app.Data_enhanceButton.Value == true
        app.Amplitude_fig_2 = log(app.Amplitude_fig_2 + 1);
    end
    if app.ApproximateButton.Value == true
        app.Amplitude_fig_2 = round(app.Amplitude_fig_2, 2);
    end
    app.Phase_fig_2 = angle(DFT_origin);
    app.Image_origin_2.ImageSource = cat(3, app.Origin_fig_2,
app.Origin_fig_2, app.Origin_fig_2);
    app.Image_ampli_2.ImageSource = cat(3, app.Amplitude_fig_2,
app.Amplitude_fig_2, app.Amplitude_fig_2);
    app.Image_phase_2.ImageSource = cat(3, app.Phase_fig_2, app.Phase_fig_2,
app.Phase_fig_2);
    app.show_3D();
end
```

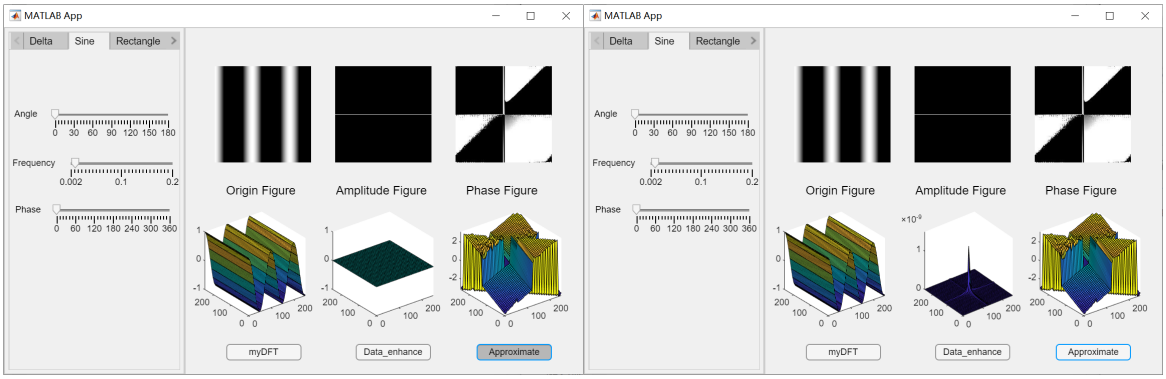
本段代码主要是利用生成的波形图像以及DFT函数将计算得到的DFT图像并通过surf显示在坐标图中并将原图显示在图片位置，本段代码主要是为了实现“与标准FFT比较”，“对DFT结果进行数据增强”，“进行模糊/准确显示”三个功能进行了$if-else$判断。

另外就是为了使得surf图显示的图像足够颜色清晰，我对于DFT的显示结果采用了显示采样的方式，以避免surf在显示的过程中由于边界线的黑色覆盖了图像的彩色。

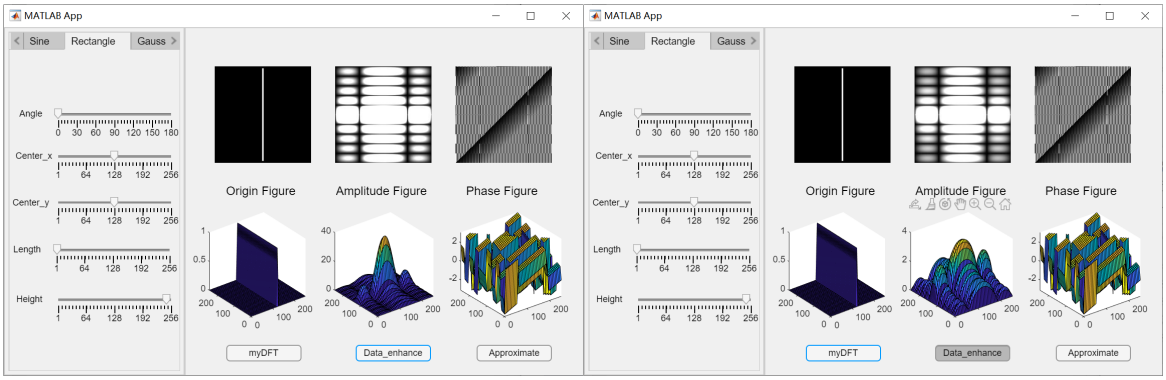
3.实验结果与分析



上述两张图片显示波形在为脉冲函数时的结果。可以看到本界面能够较好的实现要求的所有功能，并且DFt与FFT的结果基本一致，也验证了正确性。

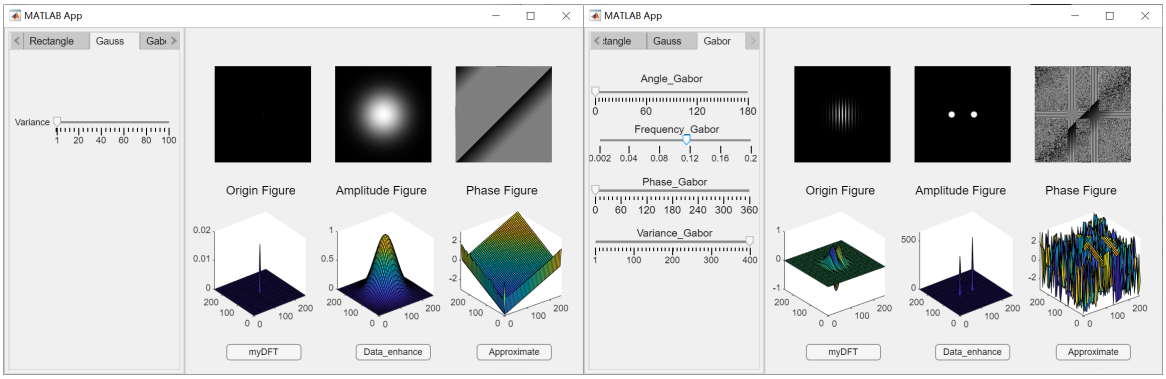


上述两张图片为显示波形在为三角函数时的结果。显示了在精确/模糊模式下DFT变换幅度谱的就显示结果差异，由于做了模糊近似，导致三角函数的DFT幅度谱结果为0，而在精确模式下却可以观察到明确的峰点出现。这也是我设计了这两种模式的初衷。



上述两张图片显示了波形在为矩形波的时候的结果。可以看见在进行数据增强后，DFT变换的幅度谱结果有着明显的变化，可以更加清晰的观察到矩形波变化的结果特征。因此我设计了是否进行数据增强这一选项。对DFT变换结果的数据增强算法依据下述公式：

$$Amplitude\_fig\_2 = log(Amplitude\_fig\_2 + 1);$$



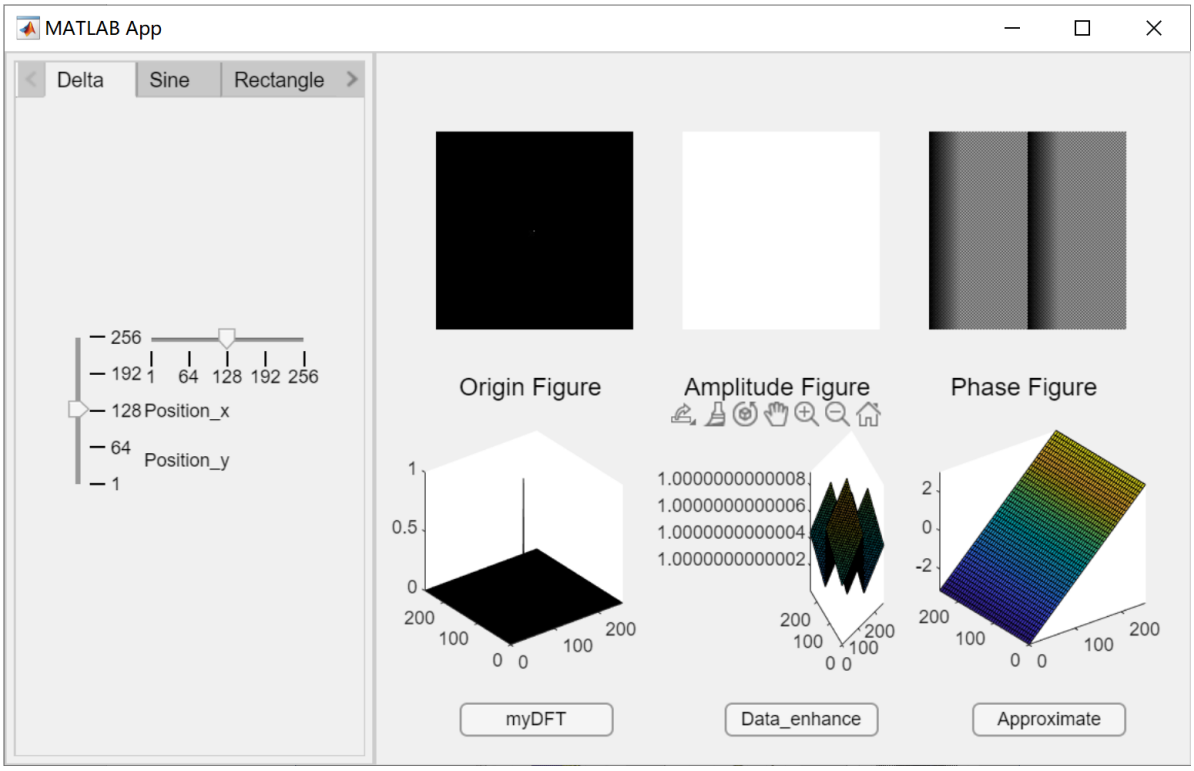
上述两张图片分别显示了在典型$Gauss$, $Gabor$信号下的DFT结果。

4.遇到的困难与解决方法

本次实验首先遇到的难点是如何设计界面布局以实现多参数可调以及六张图片的显示功能，我在对$Matlab\ app\ designer$进行了仔细研究后，还是决定采用$app\ designer$的选项卡组容器装载不同波形的控件；

之后我还遇到了surf无法显示在界面上的问题，我在查阅了大量的资料后，选择了采用将surf图片加载在坐标图中的方法实现了surf的显示；但之后，我又发现我显示出的surf图片均为纯黑色，并没有分层的彩色显示效果，在经过仔细研究后，我发现是由于$256 * 256$的图片较大，surf图边界的黑线的密集程度覆盖了色块的颜色，导致surf图片被显示为了纯黑色，为了避免这一现象，我在现实surf图片时采用了取样显示的策略，将需要显示的图片大小采样为$42 * 42$（除冲激函数之外），这样就实现了较为精确且具有彩色效果的显示结果；

但之后，我又遇到了surf图片显示的另外一个问题，$Matlab\ app\ designer$中的surf对象有很多属性无法具体设置，当坐标轴上的数据间隔很小时，会默认将数值精度显示到小数点后13位，而这样就会导致下图所示的情况：



于是我设计了如下模式将源数据近似到小数点后两位，以避免上图所示的情况。

5.收获

本次作业让我收获了Matlab的gui的编程方法，熟练掌握了Matlabapp的编程方法；同时也通过自行编写程序实验并观察的方式更加清晰地掌握了DFT的原理与性质特征，在推导DFT的过程中更是对于矩阵运算有了更深刻的理解。同时也在使用app的过程中提高了研究问题并解决问题的能力，也在设计这样一个并不算小的工程中，学习了一些管理全局变量以及封装函数的经验。

6.可能的改进方向

我认为我本次作业的完成质量还是比较可观的，有以下几个方面由于时间与能力的不足，我认为后续还可以加以提高：

1. 由于surf在app中的属性有限，不能自定义坐标轴属性，如果后续可以改善的话可以尝试用其他工具或是控件实现该功能，以避免需要设计精确与模糊模式。
2. 由于本次题目要求实现一个DFT函数，因此我没有为了进一步提高速度而简化DFT计算部分的功能而是将其封装成了一个独立的函数，如果为了提高计算速度可以在一些固定大小的基础计算矩阵的生成上进一步提高速度。

7.参考文献

无

Problem2

1.算法要点与理论原理

通过利用指纹图像的傅里叶变换结果，找到指纹脊线的方向，并绘制在原图片上。

2.算法具体实现

首先找到小块附近的大块边界点坐标，注意特殊判断边界条件。

```
for i = 0 : ceil(X / sblock) - 1
    for j = 0 : ceil(Y / sblock) - 1
        % 计算以小块为中心的大块的边际点坐标+特判
        x0 = ((i * sblock - (lblock - sblock) / 2) < 1) + ((i * sblock - (lblock - sblock) / 2) >= 1) * (i * sblock - (lblock - sblock) / 2);
        y0 = ((j * sblock - (lblock - sblock) / 2) < 1) + ((j * sblock - (lblock - sblock) / 2) >= 1) * (j * sblock - (lblock - sblock) / 2);
        x1 = ((i * sblock + (lblock - sblock) / 2 + sblock - 1) > X) * X + ((i * sblock + 23) <= X) * (i * sblock + (lblock - sblock) / 2 + sblock - 1);
        y1 = ((j * sblock + (lblock - sblock) / 2 + sblock - 1) > Y) * Y + ((j * sblock + 23) <= Y) * (j * sblock + (lblock - sblock) / 2 + sblock - 1);
        block_l = Image(x0: x1, y0: y1);
```

之后对得到的大块数据进行处理及傅里叶变换，因为傅里叶变换的结果会包含常量，因此可以先在空域对图像做减去平均值的预处理。同时，注意到待处理的图像有很多的留白区域不需要进行分析处理，可以在该部分将这些块跳过。

```
block_l = block_l - mean(mean(block_l));
if mean(mean(block_l)) <= 3
    ROI(i + 1, j + 1) = 0;
    Period(i + 1, j + 1) = 0;
    continue;
end
%imshow(block_l)

block_l = abs(fftshift(fft2(block_l)));
```

之后分析该块内的指纹脊线方向，通过课上学习到的知识与课外查询的资料知道傅里叶变换得到的两个最大值方向就是脊线的方向，而由于该值应当成对出现，故可以直接取最大的两个值进行角度与周期计算。

```
while 1
    [x, y] = find(block_l == max(max(block_l)));
    if length(x) > 1
        break;
    else
    block_l(x, y) = 0;
    end
end
ROI(i + 1, j + 1) = 1;
Direction(i + 1, j + 1) = atand((y(1) - y(2)) / (x(1) - x(2)));
Period(i + 1, j + 1) = 1 / sqrt((y(1) - y(2))^2 + (x(1) - x(2)) ^ 2);
```

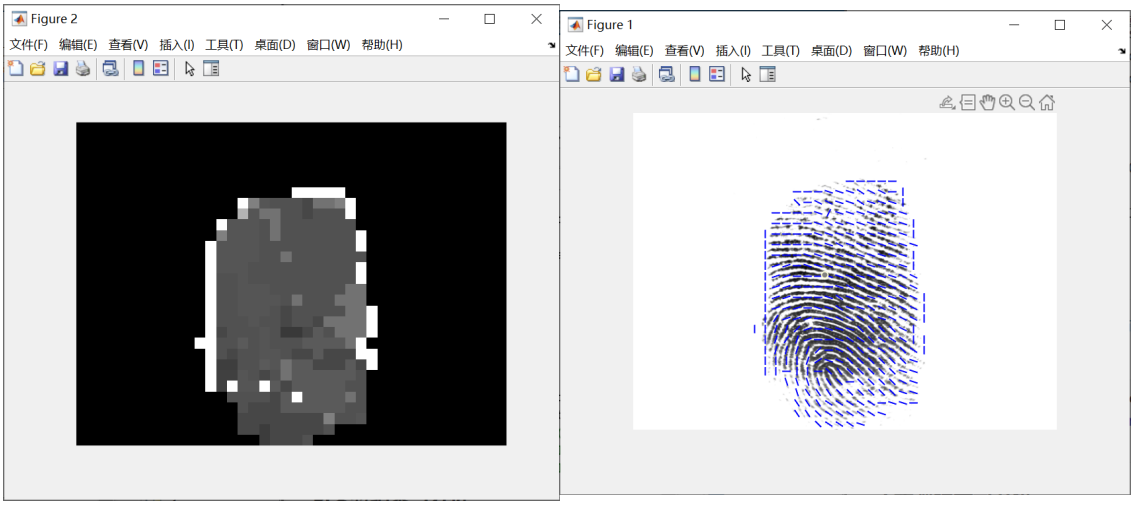
同时以上的部分也可以采用排序后结果进行计算（未在最终代码中采用，但效果也可以）

```
[block_sorted, index] = sort(block_l(:), 'descend');
[u, v] = ind2sub(size(block_l), index(1));
[m, n] = ind2sub(size(block_l), index(2));
if u == m && v == n
    Direction(i + 1, j + 1) = 0;
    Period(i + 1, j + 1) = 0;
    ROI(i + 1, j + 1) = 0;
else
    ROI(i + 1, j + 1) = 1;
    Direction(i + 1, j + 1) = atand((n - v) / (m - u));
    Period(i + 1, j + 1) = 4 / sqrt((n - v) ^ 2 + (m - u) ^ 2);
end
```

之后为了更好的显示结果，通过老师提供的显示函数指定一些参数，并对周期图进行拉伸显示处理，代码如下：

```
DrawDir(1, Direction, sblock, 'b', ROI);
Period = uint8(255 / max(max(Period)) * Period); %映射到0-255
figure(2),imshow(Period, 'InitialMagnification', 'fit');
```

3.实验结果与分析



由上述两图可知，本代码较好地完成了题目要求绘制出了题目要求的周期图像与脊线方向，只有一些边缘位置的结果由于图像的边缘有噪声的干扰出现了一些问题，但并不影响整体结果。

4.遇到的困难与解决方法

首先遇到的困难就是我在此之前并不是很清楚傅里叶变换在指纹识别中的作用与应用方式，在经过老师的介绍与和同学们的讨论之后，我才逐渐明白了工作原理与应用方式。在之后的调试过程中，由于已经多次应用matlab处理图像，我在完成本题的调试过程中还是比较顺利的，只是通过几次的输出图像与查看源数据的方式就解决了遇到的小问题。

5.收获

通过本次实验，我对傅里叶变化的性质与应用有了更清晰的认识，也掌握了傅里叶变换处理图像的一般方法与一些小技巧。

6.可能的改进方向

可以针对现在结果的边界问题加以改进算法，比如可以考虑使用一些数值分析的技巧平整边界的噪声并且使用插值法处理数据，使得结果更加准确美观。