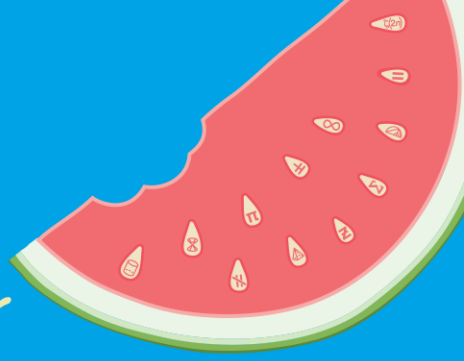
AMSI VACATIONRESEARCH SCHOLARSHIPS 2021–22

Get a taste for Research this Summer



Applicability of Discrete Fourier Transforms in Denoising Images

Sami Salem

Supervised by Sanjeeva Balasuriya

University of Adelaide

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

Contents

1 Abstract **2**

2 Introduction **2**

3 Statement of Authorship **2**

4 Methods **3**

4.1 Black and White Images 3

4.2 Fourier Transform 3

4.3 Gaussian 5

4.3.1 `fft2` of functions with domains symmetric about x and y axes 5

4.3.2 Convolution 6

4.3.3 `fft2` and `conv2` of a Gaussian 6

4.4 Edge Detection 9

5 Results **11**

5.1 Gaussian Blurring 11

5.2 Edge Detection 13

6 Discussion and Conclusion **15**

1 Abstract

This report outlines 2 techniques which can be used in a de-noising algorithm for black and white images; Gaussian blurring and edge detection. Explanation of the Discrete Fourier Transform's role in these techniques are described in detail.

2 Introduction

The main focus of this report is to show how specific properties of the Discrete Fourier Transform can be applied to Image Processing. Once the DFT of an image is calculated, it will be possible to decompose the image as the sum of many other images. These images consist of sinusoids, and they have corresponding frequencies. Then, judgement can be made as to which of these images should be removed, or which of them requires more weight. To do this, the translation of the spatial domain, (the entries of the matrix corresponding to the image), to the frequency domain is important to understand concretely - so judgement can be made on the right and wrong frequencies in the noisy image. In-order to achieve this understanding, careful construction of the Discrete Fourier Transform, (DFT), is shown.

The main application of the detailed look to the DFT is the technique of Gaussian Blurring. In this report, Gaussian Blurring refers to multiplying the DFT by a 2-dimensional Gaussian. This will affect the details of the image, (blurring it), as frequencies around 0 will be enhanced. Details of what this looks like will be described before the Gaussian is mentioned.

Another technique is Edge Detection. Edges can be detected by seeing where the derivative of the image is at it's largest. The purpose of detecting them is not the focus of the report. Rather it is presented as a section because it is a nice application of the treatment of the DFT.

3 Statement of Authorship

This report contains information which has been adapted from [Cou]. I spent time trying to understand, explain and adapt the information to my project. I then used this to explain the results in blurring and edge detection in images. The code, (shown in the appendix), was written with assistance from my supervisor, Sanjeeva Balasuriya.

4 Methods

4.1 Black and White Images

A series of assumptions will need to be stated and used throughout the project.

First we assume that an image, I, is a restriction of an integrable function, f defined on $\mathbb{R}^2$ - where $f(x, y) \to 0$ as $x \to \pm\infty$ or $y \to \pm\infty$

We consider the finite domain $D = X \times Y$ where $X, Y \in \mathbb{R}$ and $|X| = |Y|$ the image is then a matrix. Then D is a grid and $f|_D = I$ where $I(i, j) = f(D(i, j))$.

4.2 Fourier Transform

Information in this section was adapted from [Cou]

Image Processing using the Fourier Transform requires translating a noisy image to the frequency domain. The goal of this section is to associate the analytic definition of a Discrete Fourier Transform in 2 dimensions `MATLAB`'s definition. The code in the appendix also uses ideas mentioned in this section.

First, a definition will need to be stated. The two dimensional Fourier Transform is, [Alp]:

$$\hat{f}(k, m) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-2\pi i(kx+my)} dxdy$$

Where $k, m \in \mathbb{R}$.

The Inverse Fourier Transform is defined as [Alp]

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \hat{f}(k, m) e^{2\pi i(kx+my)} dkdm$$

So by the inverse Fourier Transform you can see that $\hat{f}(k, m)$ determines the contribution that $e^{2\pi i(kx+my)}$ has on f.

Now these ideas must be incorporated to images. A discrete analogue of $\hat{f}(k, m)$ is needed for computation. To do this, x and y need to be sampled. An image is defined on a sampled version of f, F, which has domain $X \times Y$ where

$$X = \{jh_1 | j \in 0, ..., N-1\}$$
$$Y = \{lh_2 | l \in 0, ..., M-1\}$$

For some $h_1, h_2 \in \mathbb{R}^+$ and $N, M \in \mathbb{N}$

An image has a positive domain, (entries in a matrix), and are equispaced with h_1 and h_2 equal to 1. In other words the domain is $X \times Y$ where X, Y range from 1 to N, M with gridpacing 1. So this choice of sampling applies to images.

The choice of sampling implies that $e^{-2\pi i(k_1 j h_1 + m_1 l h_2)} = e^{-2\pi i(k_2 j h_1 + m_2 l h_2)}$ for all j, l if $k_2 = k_1 + \frac{1}{h_1}$ and $m_2 = m_1 + \frac{1}{h_2}$. Hence $k \in [0, \frac{1}{h_1}]$ and $m \in [0, \frac{1}{h_2}]$. Then we discretise k, m to form the Discrete Fourier Transform, $\hat{F}$, of F with domain $K \times M$ where

$$K = \{\frac{n-1}{Nh_1} | n \in 1, ..., N\}$$
$$M = \{\frac{m-1}{Mh_2} | m \in 1, ..., M\}$$

In order to align the definition of the Discrete Fourier Transform, (DFT), with `MATLAB`'s definition, [MATb], we define $\hat{F} \colon K \times M \to \mathbb{R}$ as

$$\begin{aligned}\hat{F}(k,m) &= \sum_{j=1}^{j=N}\sum_{l=1}^{M} f\left((j-1)\,h_1, (l-1)\,h_2\right) e^{-2\pi i\left(\frac{k-1}{Nh_1}(j-1)h_1 + \frac{m-1}{Mh_1}(l-1)h_2\right)} \\ &= \sum_{j=1}^{j=N}\sum_{l=1}^{M} f\left((j-1)\,h_1, (l-1)\,h_2\right) e^{-2\pi i\left(\frac{k-1}{N}(j-1) + \frac{m-1}{M}(l-1)\right)}\end{aligned}$$

for $k \in 1, ..., N$ and $m \in 1, ...M$. Equivalently, for this definition $\hat{F}$ can be treated as an $N \times M$ matrix. Hence, $\hat{F}$ is now equal to `fft2(A)` in `MATLAB`, where `A(i,j)` $= f((i-1)h_1, (j-1)h_2)$

Now let F be a matrix where $F(j,l) = F((j-1)h_1, (j-2)h_2)$. Then, the definition of the Inverse Discrete Fourier Transform is, [MATc]

$$\begin{aligned}F(j,l) &= \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M} \hat{F}(k,m) e^{2\pi i\left(\frac{k-1}{Nh_1}(j-1)h_1 + \frac{m-1}{Mh_1}(l-1)h_2\right)} \\ &= \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M} \hat{F}(k,m) e^{2\pi i\left(\frac{k-1}{N}(j-1) + \frac{m-1}{M}(l-1)\right)}\end{aligned}$$

Now F is equal to `ifft2(fft2(A))` in `MATLAB`

Usually, it is only necessary to look at the real part of F, (for example if F is an image). Hence, after some algebraic manipulation, (trig identities),

$$F(j,l) = \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}\hat{A}(k,m)\cos\left(2\pi i\left(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1)\right)\right) - \hat{B}(k,m)\sin\left(2\pi i\left(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1)\right)\right)$$

Also, the periods are defined as vectors (c,d) such that

$$cos\left(2\pi i\left(\frac{k-1}{N}x+\frac{m-1}{M}y\right)\right) = cos\left(2\pi i\left(\frac{k-1}{N}(x+c)+\frac{m-1}{M}(y+c)\right)\right)$$

Analagous for $sin\left(2\pi i\left(\frac{k-1}{N}x+\frac{m-1}{M}y\right)\right)$

Hence for each $k.m$ $(c,d) = \left(\frac{N}{k-1},\frac{M}{m-1}\right)$. So for larger k and m there is smaller periods. In-other words, $\hat{F}(k,m)$ for large k and m contributes to sinusoids with small periods.

4.3 Gaussian

This section extends the definition of the DFT to calculate the DFT of a Gaussian.

4.3.1 `fft2` of functions with domains symmetric about x and y axes

We consider a function on a domain symmetric about the x,y-axes. To do this, some modifications must be made to the original discrete Fourier transform definition. Assuming the number of points are even, then the domain must be $X \times Y$ where;

$$X = \{jh_1 | j \in -N/2+1, ..., N/2\}$$
$$Y = \{lh_2 | l \in -M/2+1, ..., M/2\}$$

Then we have that

$$\hat{F}(k,m) = \sum_{j=1}^{j=N}\sum_{l=1}^{M} f(j,l)e^{-2\pi i\left(\frac{k-1}{N}\left(j-\frac{N}{2}\right)+\frac{m-1}{M}\left(l-\frac{M}{2}\right)\right)}$$
$$\hat{F}(k,m) = \sum_{j=1}^{j=N}\sum_{l=1}^{M} f(j,l)e^{-2\pi i\left(\frac{k-1}{N}j+\frac{m-1}{M}l\right)}e^{\pi i(k-1+m-1)}$$
$$\hat{F}(k,m) = \texttt{fft2(A)(k,m)} \times e^{\pi i\left(\frac{k-1}{N}+\frac{m-1}{M}\right)}$$

Then makes a matrix $mk2$ where $mk2(k,m) = e^{\pi i\left(\frac{k-1}{N}+\frac{m-1}{M}\right)}$ - then;

$$\hat{F} = \texttt{fft2(A).*mk2}$$

Also, the domain of $\hat{F}$ is from 0 to $1/h$. But we want a symmetric domain about the x,y-axes to get the blurring of the zero frequencies. Note that $\hat{F}(k-1/h, m) = \hat{F}(k, m)$, likewise for m. Thus, the frequencies $[1/2h, 1/h]$ are equivalent to $[-1/2h, 0]$. So by putting the $[1/2h, 1/h]$ frequencies before the $[0, 1/2h]$ frequencies the domain becomes $[-1/2h, 1/2h]$. This can be done by `fftshift(fft2(A).*mk2)`

4.3.2 Convolution

We define convolution by [D03]

$$f * g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) g(x - z, y - w) dz dw$$

The discrete analogue is, [MATa]

$$A * B(z, w) = \sum_{i=1}^{m} \sum_{j=1}^{n} A(i, j) B(z - i + 1, w - j + 1)$$

Another important identity is [D03]

$$\widehat{f * g}(x, y) = \hat{f}(x, y)\hat{g}(x, y)$$

Where this idea relates to

$$(\widehat{A * B})(i, j) \approx (\hat{A} \times \hat{B})(i, j) \tag{1}$$

Equation 1 is only approximate as A and B are discrete approximations of functions in 2D. So equality is not true, but the approximation should get better as $n, m \to \infty$

4.3.3 `fft2` and `conv2` of a Gaussian

Define the Gaussian $G(x, y)$ as -

$$G(x, y) = \exp\left(-\frac{x^2}{a} - \frac{y^2}{b}\right)$$

a and b correspond to the 'spread' of $G(x, y)$ about the origin in both the x and y directions respectively.

It is also true that

$$\widehat{G}(k, m) = \sqrt{ab\pi^2} \exp\left(-a\pi^2 k^2 - b\pi^2 m^2\right)$$

It is worth noting that the spread of $\hat{G}(k, m)$ is now $(w, c) = (\frac{1}{a\pi^2}, \frac{1}{b\pi^2})$.

Now the goal is to see if `fftshift(fft2(I)).*fftshift(fft2(G).*mk2)=fftshift(fft2(conv2(I,G)))` where I is the image and G is a matrix of Gaussian values. `mk2` is needed as G is defined on a symmetric domain about the origin.

This is equivalent to seeing if `fftshift(fft2(G).*mk2)` $= \hat{G}(k, m)$ where $x, y \in X, Y$, which are sampled domains.

The accuracy should depend on how large X and Y are - which depends on the choice of a and b.

Figure 1 shows that the error decreases for smaller dx, for a constant domain. The domain was -80 to 80 on each side, with a, b terms equal to 2.

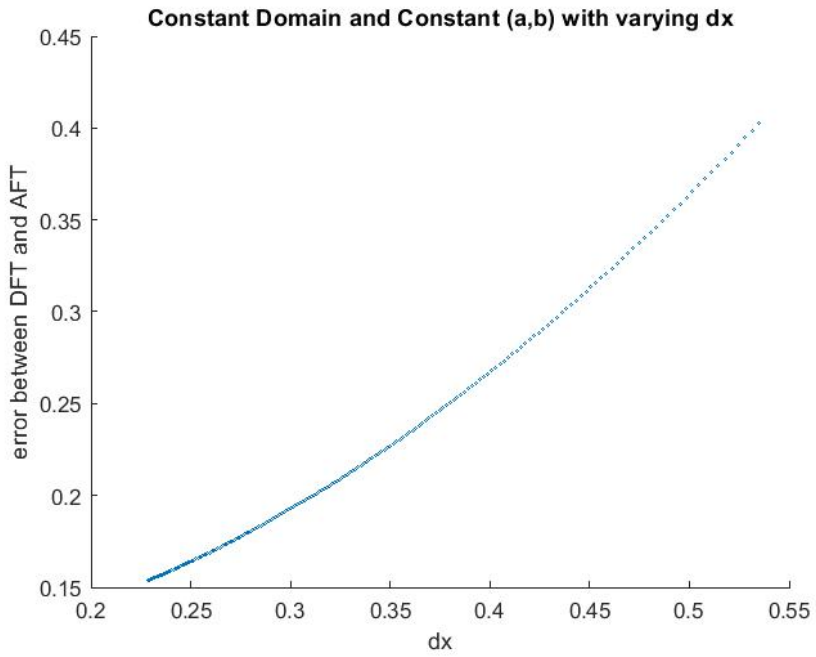


Figure 1: dx vs error of discrete approximation of FT of Gaussian vs DFT of Gaussian

Figure 2 will look at keeping the number of points constant, (500×500 Gaussian), with changing dx which will change the domain.

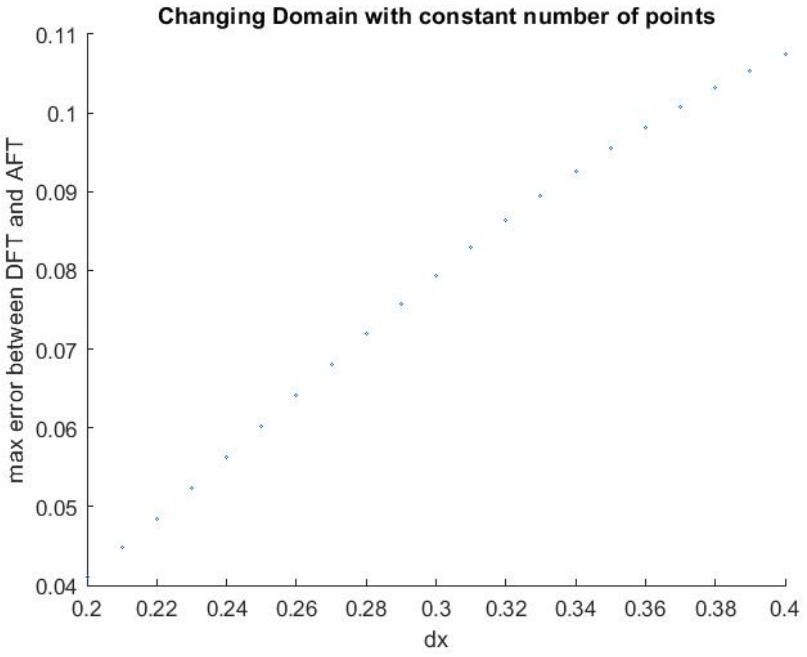


Figure 2: dx vs error but now the domain is changing

This means that once the (a, b) terms have been chosen, care must be taken to first find a domain which is large and encompasses the Gaussian - and then the dx should be small. This is clear from Figure 2 as the larger dx, (corresponding to larger error), results in a smaller domain - thus less encompassing of the Gaussian. Also Figure 1 shows for constant domain, smaller dx gives less error.

When deciding on the a and b terms in the Gaussian, it is important to first look at what basis functions corresponding to different frequencies look like. By the previous sections,

$$I(j,l) = \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}\hat{A}(k,m)cos\left(2\pi i\left(\frac{k-1}{N}jh_1 + \frac{m-1}{M}lh_2\right)\right) - \hat{B}(k,m)sin\left(2\pi i\left(\frac{k-1}{N}jh_1 + \frac{m-1}{M}lh_2\right)\right)$$

Where I is the image and $\hat{I} = \hat{A} + i\hat{B}$.

This can be simplified in matrix form by saying that

$$I = \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}\hat{A}(k,m)cos\left(2\pi i\left(\frac{k-1}{N}.*X + \frac{m-1}{M}.*Y\right)\right) - \hat{B}(k,m)sin\left(2\pi i\left(\frac{k-1}{N}.*X + \frac{m-1}{M}.*Y\right)\right)$$

Where $X = \{1, 2, ..., N\}$ and $Y = \{1, 2, ..., M\}$, as I is an image. To visualise what is going on, we see what $\hat{A}(k,m)cos\left(2\pi i\left(\frac{k-1}{N}.*X + \frac{m-1}{M}.*Y\right)\right) - \hat{B}(k,m)sin\left(2\pi i\left(\frac{k-1}{N}.*X + \frac{m-1}{M}.*Y\right)\right)$ looks like for specific k and m terms. To compare with the frequency domain, (where frequencies are measured as the constant times the 2π term), we look at sinusoids with $f1 = (k-1)/N, f2 = (m-1)/M$, (see Figure 3).

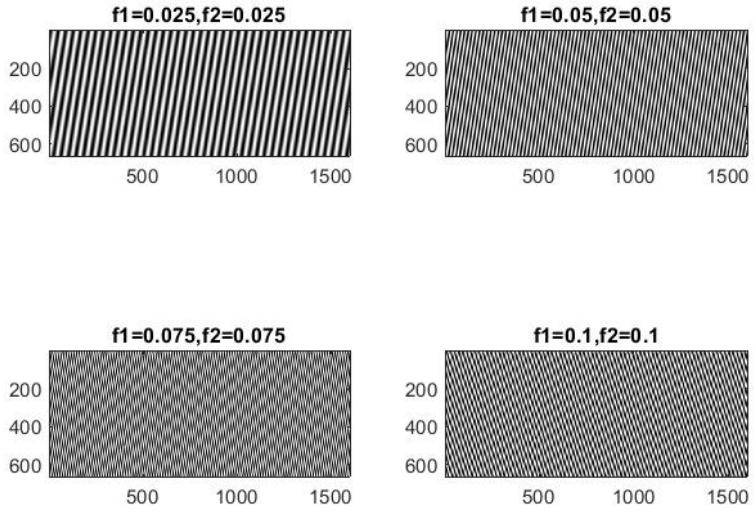


Figure 3: Real basis functions of the DFT

It is clear from Figure 3 that in order to blur an image it would be best to focus on frequencies less than around $f1, f2$ around 0.025 to 0.05. As the frequency domain of an image is from -0.5 to 0.5, then the width of the DFT of the Gausian should be around a tenth or a twentieth of the width of it's frequency domain, (which depends on the choice of h_1 and h_2 values). This means the DFT of the Gaussian, (which is the same size as the image FFT), will then amplify the frequencies around 0.025 and 0.05.

4.4 Edge Detection

Edges in an image can be detected by larger local changes in the image. In other words; where the partial derivatives are larger. This is done by using the *sobel operator*, which calculates the magnitude of the gradient at each point in the image, [O17]. The edges should have a larger gradient and should thus appear whiter.

Finding the partial derivative of the image can be complicated numerically. However, finding the Fourier transform of the derivative and then inverting is more achievable.

$$\begin{aligned}\hat{f}_x(k,m) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_x(x,y)e^{-2\pi i(kx+my)}dxdy \\ \hat{f}_x(k,m) &= \int_{-\infty}^{\infty} -2\pi ik \int_{-\infty}^{\infty} f(x,y)e^{-2\pi i(kx+my)}dxdy \\ &= -2\pi ik\hat{f}(k,m)\end{aligned}$$

This is because calculating the inner integral, $\int_{-\infty}^{\infty} f_x(x,y)e^{-2\pi i(kx+my)}dx$, is the same as calculating

$$\int_{-\infty}^{\infty} f^{'}(x)e^{-2\pi ikx}dx = -2\pi ik\hat{f}(k)$$

, [Alp].

So inverting $-2\pi ik\hat{f}(k,m)$ gives $f_x(k,m)$. Analogously, $-2\pi im\hat{f}(k,m)$ gives $f_y(k,m)$.

In `MATLAB` this is equivalent to `ifft2(-2πik.*fft2(A))` gives a partial derivative of entries in A.

For image processing purposes, edge detection should be done on a blurred image to remove the noisy pixels being detected.

5 Results

5.1 Gaussian Blurring

First consider the Figures 4 and 5. They are black and white images with noise added to them,

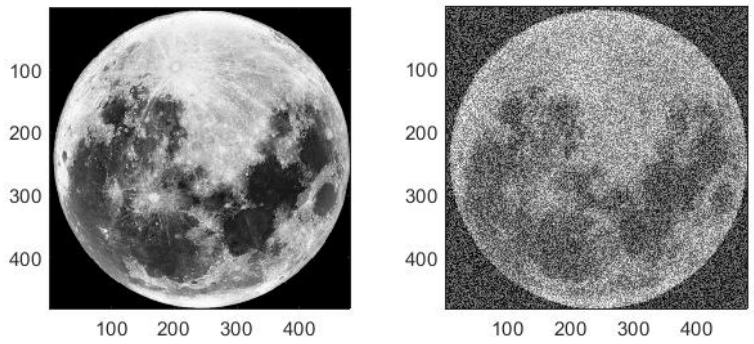


Figure 4: Picture of moon on it's own vs picture with noise added to each pixel

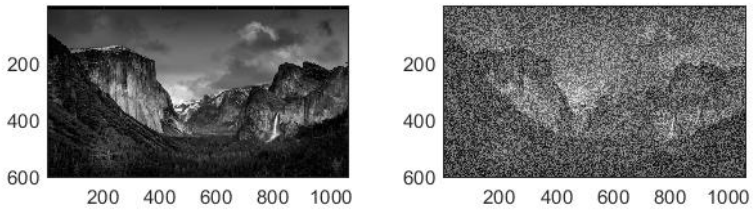


Figure 5: Picture of mountain on it's own vs picture with noise added to each pixel

Figures 6 and 7, show what happens when Figures 4 and 5 are Gaussian blurred. The panel shows a variety of choices for (w, c), (the width of the DFT of the Gaussian).

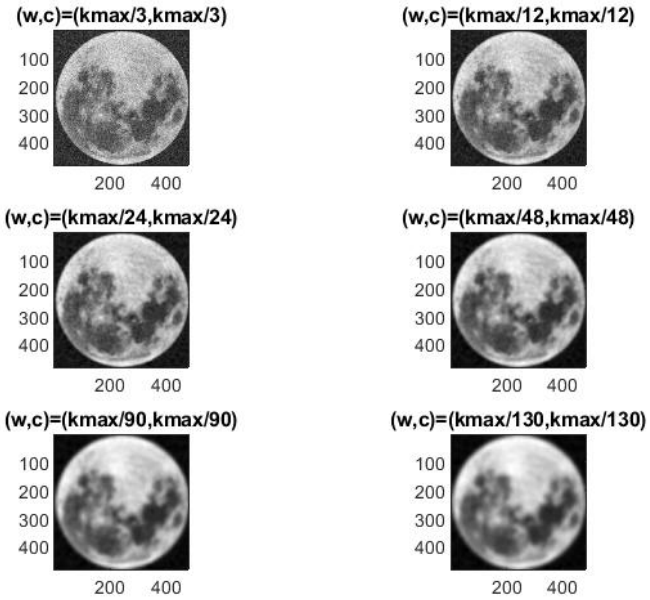


Figure 6: Noisy moon with Gaussian blurring - via DFT with various (w,c) values

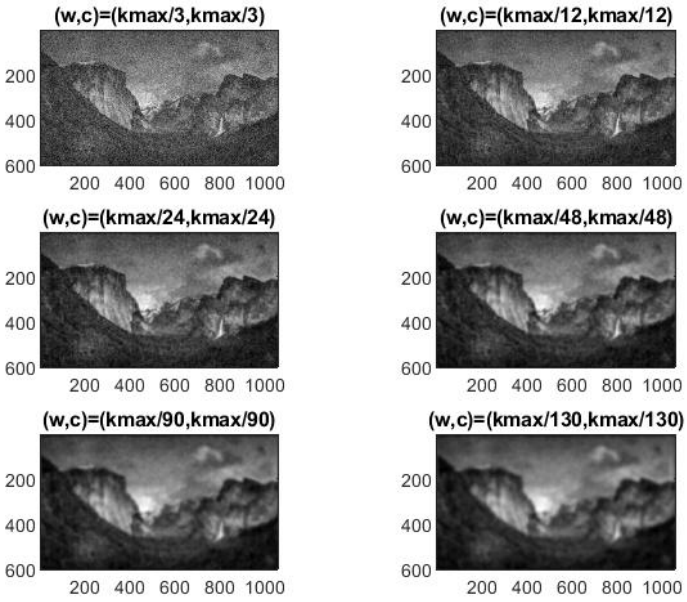


Figure 7: Noisy mountain with Gaussian blurring - via DFT with various (w,c) values

Figures 6 and 7 show that the image getes blurrier and less defined as (w, c) gets smaller.

5.2 Edge Detection

Figures 8 and 9 show edge detection being applied to Figures 6 and 7.

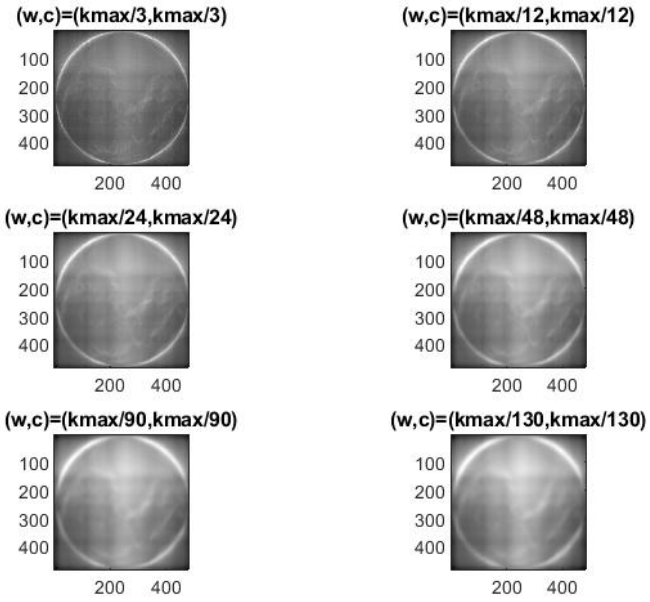


Figure 8: Noisy moon with Gaussian blurr from Figure 6 - with edge detection

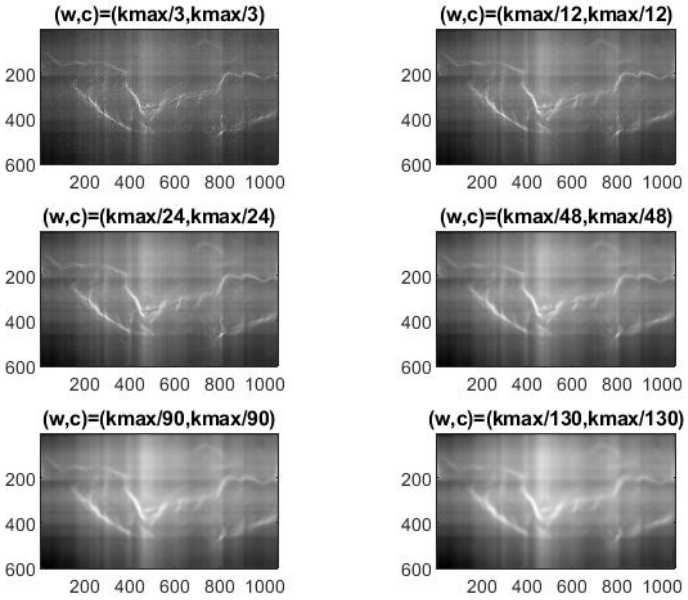


Figure 9: Noisy mountain with Gaussian blurr from Figure 7 - with edge detection

Both Figures 8 and 9 show that there are white pixels approximately where the edges are in Figures 6 and 7. The accuracy gets better as (w, c) gets smaller.

6 Discussion and Conclusion

The definition of a Discrete Fourier Transform has proven to be useful in image processing. The images with noise have been blurred using ideas that were generated from the Methods section. The results were fairly consistent - namely that focusing on smaller frequencies create blurrier images. The edge detection was also working well compared to the theory. The derivatives of the image corresponded with edges, and the DFT was used to find the derivatives.

The Results section shows that the choice of (w, c), (spread of DFT of Gaussian), affects the blurring of a noisy image. Choosing $(w, c) = (kmax/a, kmax/a)$ where a ranges from roughly 10 to 100, gives a balance between blurring and detail, and thus seems to give the best results. Choosing smaller values of (w, c) seems to give too much blurring, where-as larger values are not effective in removing noise.

The edge detection images also show some of it's own kind of noise. The grey horizontal and vertical streaks are distracting, and is an unanticipated effect of the code. Also, the accuracy of the edge detection increases as the blurring decreases, which is no surprise as edges are less defined when images are blurred. Nevertheless, detecting the edges could come to use by extending the algorithm to try and see where the edges in the blurred images are and making them more pronounced.

Also, comparing Gaussian blurring via convolution and the DFT method was difficult - because the discrete approximation of the Gaussian yields error for the convolution.

Appendix This code generated the blurring and edge detection images.

```
 close all

A2=(double(imread('mountains.png')));

%Image could be RGB so could have 3 components for each entry
[n,m,z]=size(A2);

%If n or m is odd then let n = n-1 or m=m-1%

%Convert A2 to grayscale image%
BA2=A2(1:n,1:m);

%Add the random noise to the image%
A2=BA2+255*rand(n,m);

%Calculate the DFT of the image%
FA2=(fft2(A2));
%------------------frequency domain--------------------------------------%

%Spatial domain of the Gaussian%
dx=0.25;
dy=0.25;
X1=-(n-1)*dx/2:dx:(n-1)*dx/2;
X2=-(m-1)*dx/2:dy:(m-1)*dx/2;
[X,Y]=meshgrid(X2,X1);

n=length(X1);
m=length(X2);

%frequency domain of Gaussian%
kmax=1/(2*dx);
k=linspace(-kmax,kmax,n);
k2=linspace(-kmax,kmax,m);
[K1,K2]=meshgrid(k2,k);
```

```
%--------------------------Gaussian---------------------------------------%

%The mk2 matrix to make the DFT in matlab compatible with negative/positive domain%
mk2=ones(n,m);
for i=1:n
    for j=1:m
        mk2(i,j)=exp(pi*(1j)*((i-1)+(j-1)));
    end
end


%Calculating various Gaussians with varying a,b terms%
a=3/(kmax*pi^2);b=3/(kmax*pi^2);
f3=@(x,y) exp(-(x.^2)./a-(y.^2)./b);
V2=f3(X,Y);

Z2=(fftshift(fft2(V2).*mk2))*dx*dy;

FV2=(sqrt(pi*a)*sqrt(pi*b))*exp(-pi^2*K1.^2*a-pi^2*K2.^2*b);

a=12/(kmax*pi^2);b=12/(kmax*pi^2);
f3=@(x,y) exp(-(x.^2)./a-(y.^2)./b);
[X,Y]=meshgrid(X2,X1);
V22=f3(X,Y);

Z22=(fftshift(fft2(V22).*mk2))*dx*dy;

FV22=(sqrt(pi*a)*sqrt(pi*b))*exp(-pi^2*K1.^2*a-pi^2*K2.^2*b);

a=24/(kmax*pi^2);b=24/(kmax*pi^2);
f3=@(x,y) exp(-(x.^2)./a-(y.^2)./b);
[X,Y]=meshgrid(X2,X1);
V23=f3(X,Y);

Z23=(fftshift(fft2(V23).*mk2))*dx*dy;

FV23=(sqrt(pi*a)*sqrt(pi*b))*exp(-pi^2*K1.^2*a-pi^2*K2.^2*b);
```

```
a=48/(kmax*pi^2);b=48/(kmax*pi^2);
f3=@(x,y) exp(-(x.^2)./a-(y.^2)./b);
[X,Y]=meshgrid(X2,X1);
V24=f3(X,Y);

Z24=(fftshift(fft2(V24).*mk2))*dx*dy;

FV24=(sqrt(pi*a)*sqrt(pi*b))*exp(-pi^2*K1.^2*a-pi^2*K2.^2*b);

a=90/(kmax*pi^2);b=90/(kmax*pi^2);
f3=@(x,y) exp(-(x.^2)./a-(y.^2)./b);
[X,Y]=meshgrid(X2,X1);
V25=f3(X,Y);

Z25=(fftshift(fft2(V25).*mk2))*dx*dy;

FV25=(sqrt(pi*a)*sqrt(pi*b))*exp(-pi^2*K1.^2*a-pi^2*K2.^2*b);

a=130/(kmax*pi^2);b=130/(kmax*pi^2);
f3=@(x,y) exp(-(x.^2)./a-(y.^2)./b);
[X,Y]=meshgrid(X2,X1);
V26=f3(X,Y);

Z26=(fftshift(fft2(V26).*mk2))*dx*dy;

FV26=(sqrt(pi*a)*sqrt(pi*b))*exp(-pi^2*K1.^2*a-pi^2*K2.^2*b);

figure(80)
surf(X,Y,V2);

figure(8)
surf(K2,K1,FV2)
title('Fourier transform, (using equation),')
figure(9)
surf(K2,K1,real(Z2))
title('Fourier transform, (using MATLAB)')
```

```
%calculating the errors between the DFT using MATLAB and the analytical DFT
%The error gets larger as the standard deviation, (a,b), gets larger for a
%constant domain%

error21=max(max(abs(FV2-real(Z2))));
error22=max(max(abs(FV22-real(Z22))));
error23=max(max(abs(FV23-real(Z23))));
error24=max(max(abs(FV24-real(Z24))));
error25=max(max(abs(FV25-real(Z25))));
error26=max(max(abs(FV26-real(Z26))));

error = [error21;error22;error23;error24;error25;error26]

%--------------------------blurring--------------------------------------%

%Performing the Gaussian Blurring%
CF1=(Z2).*(fftshift(FA2));
CF2=(Z22).*(fftshift(FA2));
CF3=(Z23).*(fftshift(FA2));
CF4=(Z24).*(fftshift(FA2));
CF5=(Z25).*(fftshift(FA2));
CF6=(Z26).*(fftshift(FA2));

CIF=abs(ifft2(ifftshift(CF1)));

CIF2=abs(ifft2(ifftshift(CF2)));

CIF3=abs(ifft2(ifftshift(CF3)));

CIF4=abs(ifft2(ifftshift(CF4)));

CIF5=abs(ifft2(ifftshift(CF5)));

CIF6=abs(ifft2(ifftshift(CF6)));

%plotting the various Gaussian blurred images - via DFT%
```

```
figure(5)
subplot(3,2,1),
image((CIF),'CDataMapping','scaled'),title('(w,c)=(kmax/3,kmax/3)');
axis image
colormap gray
subplot(3,2,2),
image(CIF2,'CDataMapping','scaled'),title('(w,c)=(kmax/12,kmax/12)');
axis image
colormap gray
subplot(3,2,3),
image((CIF3),'CDataMapping','scaled'),title('(w,c)=(kmax/24,kmax/24)');
axis image
colormap gray
subplot(3,2,4),
image(CIF4,'CDataMapping','scaled'),title('(w,c)=(kmax/48,kmax/48)');
axis image
colormap gray
subplot(3,2,5),
image((CIF5),'CDataMapping','scaled'),title('(w,c)=(kmax/90,kmax/90)');
axis image
colormap gray
subplot(3,2,6),
image(CIF6,'CDataMapping','scaled'),title('(w,c)=(kmax/130,kmax/130)');
axis image
colormap gray



%------------------------------edges---------------------------------------%

%Performing edge detection for each of the Gaussian blurred images from
%before%

v1=linspace(-1/(2*dx),1/(2*dx),n);
v2=linspace(-1/(2*dy),1/(2*dy),m);
[VV1,VV2]=meshgrid(v2,v1);
```

```
CF2x=((-2*pi*(1i))^1*(VV1).^1).*(fftshift(CF1)./(dx*dy));
CF2y=((-2*pi*(1i))^1*(VV2).^1).*(fftshift(CF1)./(dx*dy));
ICFx=(ifft2(CF2x));
ICFy=(ifft2(CF2y));
ICF=sqrt(abs(ICFx).^2+abs(ICFy).^2);
ICF=rescale(ICF,0,255);

v1=linspace(-1/(2*dx),1/(2*dx),n);
v2=linspace(-1/(2*dy),1/(2*dy),m);
[VV1,VV2]=meshgrid(v2,v1);
CF2x=((-2*pi*(1i))^1*(VV1).^1).*(fftshift(CF2)./(dx*dy));
CF2y=((-2*pi*(1i))^1*(VV2).^1).*(fftshift(CF2)./(dx*dy));
ICFx=(ifft2(CF2x));
ICFy=(ifft2(CF2y));
ICF2=sqrt(abs(ICFx).^2+abs(ICFy).^2);
ICF2=rescale(ICF2,0,255);

v1=linspace(-1/(2*dx),1/(2*dx),n);
v2=linspace(-1/(2*dy),1/(2*dy),m);
[VV1,VV2]=meshgrid(v2,v1);
CF2x=((-2*pi*(1i))^1*(VV1).^1).*(fftshift(CF3)./(dx*dy));
CF2y=((-2*pi*(1i))^1*(VV2).^1).*(fftshift(CF3)./(dx*dy));
ICFx=(ifft2(CF2x));
ICFy=(ifft2(CF2y));
ICF3=sqrt(abs(ICFx).^2+abs(ICFy).^2);
ICF3=rescale(ICF3,0,255);

v1=linspace(-1/(2*dx),1/(2*dx),n);
v2=linspace(-1/(2*dy),1/(2*dy),m);
[VV1,VV2]=meshgrid(v2,v1);
CF2x=((-2*pi*(1i))^1*(VV1).^1).*(fftshift(CF4)./(dx*dy));
CF2y=((-2*pi*(1i))^1*(VV2).^1).*(fftshift(CF4)./(dx*dy));
ICFx=(ifft2(CF2x));
ICFy=(ifft2(CF2y));
ICF4=sqrt(abs(ICFx).^2+abs(ICFy).^2);
ICF4=rescale(ICF4,0,255);
```

```
v1=linspace(-1/(2*dx),1/(2*dx),n);
v2=linspace(-1/(2*dy),1/(2*dy),m);
[VV1,VV2]=meshgrid(v2,v1);
CF2x=((-2*pi*(1i))^1*(VV1).^1).*(fftshift(CF5)./(dx*dy));
CF2y=((-2*pi*(1i))^1*(VV2).^1).*(fftshift(CF5)./(dx*dy));
ICFx=(ifft2(CF2x));
ICFy=(ifft2(CF2y));
ICF5=sqrt(abs(ICFx).^2+abs(ICFy).^2);
ICF5=rescale(ICF5,0,255);

v1=linspace(-1/(2*dx),1/(2*dx),n);
v2=linspace(-1/(2*dy),1/(2*dy),m);
[VV1,VV2]=meshgrid(v2,v1);
CF2x=((-2*pi*(1i))^1*(VV1).^1).*(fftshift(CF6)./(dx*dy));
CF2y=((-2*pi*(1i))^1*(VV2).^1).*(fftshift(CF6)./(dx*dy));
ICFx=(ifft2(CF2x));
ICFy=(ifft2(CF2y));
ICF6=sqrt(abs(ICFx).^2+abs(ICFy).^2);
ICF6=rescale(ICF6,0,255);

figure(11)
subplot(3,2,1)
image(ICF,'CDataMapping','scaled'),axis image,colormap gray, title('(w,c)=(kmax/3,kmax/3)');
subplot(3,2,2)
image(ICF2,'CDataMapping','scaled'),axis image,colormap gray, title('(w,c)=(kmax/12,kmax/12)');
subplot(3,2,3)
image(ICF3,'CDataMapping','scaled'),axis image,colormap gray, title('(w,c)=(kmax/24,kmax/24)');
subplot(3,2,4)
image(ICF4,'CDataMapping','scaled'),axis image,colormap gray, title('(w,c)=(kmax/48,kmax/48)');
subplot(3,2,5)
image(ICF5,'CDataMapping','scaled'),axis image,colormap gray, title('(w,c)=(kmax/90,kmax/90)');
subplot(3,2,6)
image(ICF6,'CDataMapping','scaled'),axis image,colormap gray, title('(w,c)=(kmax/130,kmax/130)');



%Plotting the original image and the blurred image%
```

```
figure(2221211)
subplot(1,2,2), image(A2,'CDataMapping','scaled'),axis image,colormap gray
subplot(1,2,1), image(BA2,'CDataMapping','scaled'),axis image,colormap gray
```

This code was used to calculate the dx vs error images. This code is for the constant domain

```
function [dx,merr,K1,K2,dfX,afX]=sami_gaussian2d_sderror(a)

for i=1:length(a)
    dx(i)=160/(a(i)-1);
    X=-80:dx(i):80;
    Y=X;
    v=length(X);
    [X,Y]=meshgrid(X,Y);
    k=linspace(-1/(2*dx(i)),1/(2*dx(i)),v);
    [K1,K2]=meshgrid(k,k);
    b=2;
    eX=exp(-X.^2./b-Y.^2./b);
    mm=ones(v,v);
    for o=1:v
        for t=1:v
            mm(o,t)=exp(pi*(1j)*(o-1))*exp(pi*(1j)*(t-1));
        end
    end
    dfX=real(fftshift(fft2(eX).*mm))*dx(i)*dx(i);
    afX=b*pi*exp(-b*pi^2*K1.^2-b*pi^2*K2.^2);
    merr(i)=max(max(abs(dfX-afX)));
end
```

Now for the changing domain

```
    function [dx,merr,K1,K2,dd,aa]=sami_gaussian2d_sdvserror(a)

for i=1:length(a)
    dx(i)=a(i);
    X=-a(i)*(500-1)/2:a(i):a(i)*(500-1)/2;
    Y=X;
```

```
    v=length(X);
    [X,Y]=meshgrid(X,Y);
    k=linspace(-1/(2*dx(i)),1/(2*dx(i)),v);
    [K1,K2]=meshgrid(k,k);
    b=0.05;
    eX=exp(-X.^2./b-Y.^2./b);
    mk2=ones(v,v);
    for o=1:v
        for t=1:v
            mk2(o,t)=exp(pi*(1j)*((o-1)+(t-1)));
        end
    end
    dfX=real(fftshift(fft2(eX).*mk2))*dx(i)*dx(i);
    dd=dfX;
    afX=b*pi*exp(-b*pi^2*K1.^2-b*pi^2*K2.^2);
    aa=afX;
    merr(i)=max(max(abs(dfX-afX)));
end
```

This code produced the image relating to the basis functions of the image,

```
    X=linspace(1,n,665);
Y=linspace(1,m,1600);

[X,Y]=meshgrid(Y,X);

A=20; B=300;

f1=A*cos(2*pi*(0.999)*X+((0.999)*Y))-B*sin(2*pi*(0.999)*X+(0.999)*Y);

f2=A*cos(2*pi*(0.05)*X+(0.05*Y))-B*sin(2*pi*(0.05)*X+(0.05)*Y);

f3=A*cos(2*pi*(0.3)*X+(0.075*Y))-B*sin(2*pi*(0.075)*X+(0.075)*Y);

f4=A*cos(2*pi*(0.4)*X+(0.1*Y))-B*sin(2*pi*(0.1)*X+(0.1)*Y);

title('A(r,v)=20,B(r,v)=300')
subplot(2,2,1)
```

```
image(f1,'CDataMapping','scaled'), colormap gray, axis image, title('f1=0.025,f2=0.025')
subplot(2,2,2)
image(f2,'CDataMapping','scaled'), colormap gray, axis image, title('f1=0.05,f2=0.05')
subplot(2,2,3)
image(f3,'CDataMapping','scaled'), colormap gray, axis image, title('f1=0.075,f2=0.075')
subplot(2,2,4)
image(f4,'CDataMapping','scaled'), colormap gray, axis image, title('f1=0.1,f2=0.1')
```

This is the derivation of the real part of the DFT,

$$
\begin{aligned}
F(j,l) &= \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}\hat{F}(k,m)e^{2\pi i(\frac{k-1}{Nh_1}(j-1)h_1+\frac{m-1}{Mh_1}(l-1)h_2)} \\
&= \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}(\hat{A}(k,m)+i\hat{B}(k,m))(\cos(2\pi(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1))+i\sin(2\pi(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1)) \\
&= \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}\hat{A}(k,m)\cos(2\pi(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1))+i\hat{A}(k,m)\sin(2\pi(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1)) \\
&+ \frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}i\hat{B}(k,m)\cos(2\pi(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1))-\hat{B}(k,m)\sin(2\pi(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1))
\end{aligned}
$$

So the real part is

$$
\frac{1}{MN}\sum_{k=1}^{k=N}\sum_{m=1}^{M}\hat{A}(k,m)\cos(2\pi\left(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1)\right)-\hat{B}(k,m)\sin(2\pi\left(\frac{k-1}{N}(j-1)+\frac{m-1}{M}(l-1)\right)
$$

Bibliography

[D03] Bracewell D. *Fourier Analysis and Imaging*. Springer, 2003. ISBN: 978- 1-4613-4738-5.

[O17] Gonazalez CI Melin P Castro JR Castillo O. *Edge Detection Based on Generalized Fuzzy Type-2 Logic*. Studies in Computational Intelligence. Springer, 2017. ISBN: 978-3-319-53993-5.

[Alp] Wolfram Alpha. *Fourier Transform*. URL: https://mathworld.wolfram.com/FourierTransform.html. (accessed 16:02:2022).

[Cou] MIT Open Courseware. *Chapter 6 Fourier Analysis*. URL: https://ocw.mit.edu/courses/mathematics/18-330-introduction-to-numerical-analysis-spring-2012/lecture-notes/MIT18_330S12_Chapter6.pdf. (accessed 14:02:2022).

[MATa] MATLAB. *2-D convolution*. URL: https://au.mathworks.com/help/matlab/ref/conv2.html#bvgtfv6. (accessed 19:02:2022).

[MATb] MATLAB. *fft2*. URL: https://au.mathworks.com/help/matlab/ref/fft2.html. (accessed 14:02:2022).

[MATc] MATLAB. *ifft2*. URL: https://au.mathworks.com/help/matlab/ref/ifft2.html. (accessed 26:02:2022).