DFTs

Thursday, July 12, 2018 3:08 PM

Discrete Fourier Transforms (DFTs) are required to process Fourier transforms on a computer since most problems of interest are difficult or impossible to write analytically. Consequently, these problems require numerical methods. DFTs let us use much of the theory we have developed so far, but there are certain limitations to DFTs that must be understood to ensure accurate numerical results.

This treatment follows Voelz's Computational Fourier Optics

Recall our definition of the Fourier transform

$$F(\xi) = \int_{-\infty}^{\infty} f(x) \exp[-i2\pi \xi x]\, dx$$

and its inverse

$$f(x) = \int_{-\infty}^{\infty} F(\xi)\, e^{i2\pi\xi x}\, d\xi$$

We want to rewrite these integrals as discrete sums.

$$F(\xi_m) = \mathcal{F}\{ f(x_N) \} = \sum_{N=-\infty}^{\infty} f(x_N) \exp(-i2\pi \xi_m x_N)\,(x_{N+1} - x_N)$$

where $m = -\infty \ldots \infty$ and N and m are integers

In the computer, we can only deal with a finite number of samples N (We'll only use even values of N because many of the fast algorithms for calculating DFTs require this.)

The spacing between samples in the spatial domain is X_s. This means the individual samples $x_N = N X_s$. In the frequency domain, the spacing between samples is $\frac{1}{N X_s}$, so $\xi_m = \frac{m}{N X_s}$.

Incorporating these values gives

$$F\left(\frac{m}{NX_s}\right) = \mathcal{F}\{f(nX_s)\} = X_s \sum_{n=-N/2}^{N/2-1} f(nX_s)\exp(-i2\pi mn/N)$$

where $m = -\frac{N}{2}, -\frac{N}{2}+1 \ldots \frac{N}{2}-1$

In computer programs we would normally define these discretely sampled functions as an array. Programming languages usually start their counting for array at either 0 or 1. C and C++ use 0 and Matlab uses 1. We'll use the 1 index since many of you use Matlab for the homeworks.

$g[1], g[2], g[3] \ldots g[N]$ g is an N dimensional vector or a $N \times 1$ dimensional matrix.

We can think of our sampled function $f(nX_s)$ as

$f[-N/2], f[-N/2+1] \ldots f[N/2-1]$ with the understanding that each element of the vector holds a sample of the function with spacing X_s. To map the vector to the Matlab form

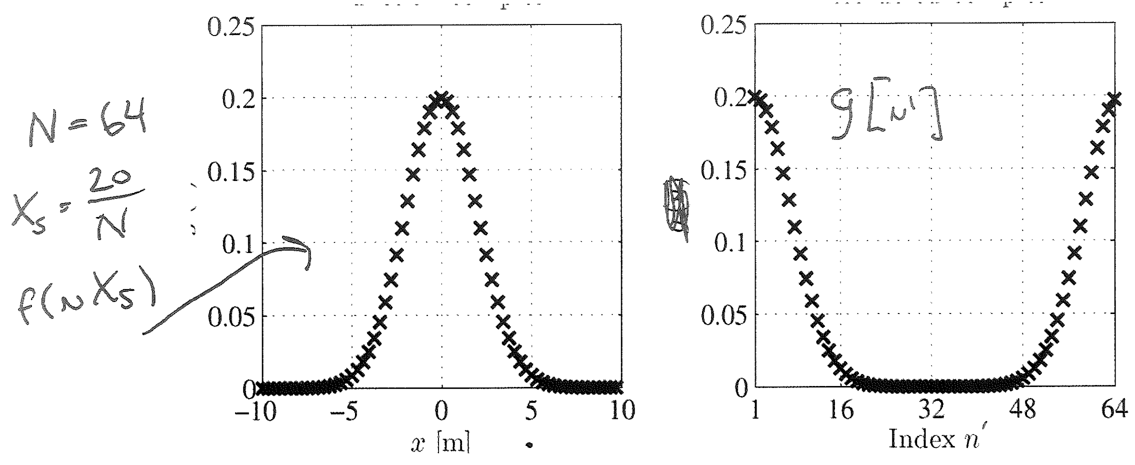
$g[n'] = f((n' + \frac{N}{2})X_s)$ or $n' = 1, 2 \ldots \frac{N}{2}+1$

$g[n'] = f((n' - N - 2)X_s$ or $n' = \frac{N}{2}+2, \frac{N}{2}+3 \ldots N$

The Fourier transform can now be written in terms of these properly indexed vectors

$$G[m'] = X_s \sum_{n'=1}^{N} g[n'] \exp\left[-i2\pi(m'-1)(n'-1)/N\right]$$

where $m' = 1, 2, 3 \ldots N$ (N−1) and $G[\,]$ is a vector holding the Fourier transform values



Screen clipping taken: 10/31/2019 7:09 PM

This is the same as shifting the function to the left and wrapping the values around on the right.

The inverse transform looks very similar

$$g[n'] = \frac{1}{N X_s} \sum_{m'=1}^{N} G[m'] \exp\left[i2\pi(m'-1)(n'-1)/N\right]$$

with $n' = 1, 2, 3 \ldots N$.

Computationally, Matlab assumes $X_s = 1$ unit. To get the correct units, it's up to you to properly scale the output of Matlab's fft and ifft routines.

```
G = fftshift(fft(fftshift(g))) * delta;
```
FORWARD TRANSFORM

Screen clipping taken: 10/31/2019 7:42 PM

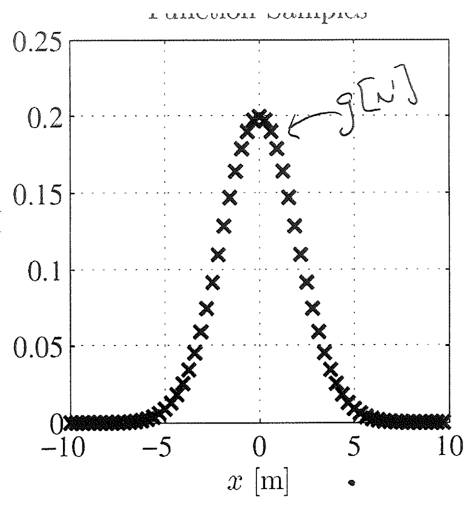
fftshift does the shifting described above and delta $= X_s$ in our notation.

```
g = ifftshift(ifft(ifftshift(G))) ...
    * length(G) * delta_f;
```

Screen clipping taken: 10/31/2019 7:43 PM

ifftshift does the shifting

length(G) = N
delta_f = $\frac{1}{N X_s}$ in our notation.



Screen clipping taken: 10/31/2019 7:49 PM

Example

$$g(x) = 0.2 \, \text{Gaus}\left(\frac{x}{10}\right)$$

$N = 64$ samples
covering a region from −10m to 10m

$$X_s = \frac{20\text{m}}{64} = 0.3125\text{ m} \quad \text{(Sample spacing)}$$

$g[1] = 0.2 \, \text{Gaus}(0)$ SHIFTED
$g[2] = 0.2 \, \text{Gaus}(0.3125)$ DISCRETE
$g(3] = 0.2 \, \text{Gaus}(0.6250)$ VALUES

The spacing in the Fourier domain is

$$\xi_{m+1} - \xi_m = \frac{1}{N X_s} = \frac{1}{64(0.3125\text{m})} = 0.05 \frac{\text{cyc}}{\text{m}}$$

SUMMARY

FORWARD TRANSFORM

$$G[m'] = X_s \sum_{N'=1}^{N} g[N'] \exp\left[-i2\pi (m'-1)(N'-1)/N\right]$$

where $m' = 1, 2, 3 \ldots N$

INVERSE TRANSFORM

$$g[N'] = \frac{1}{N X_s} \sum_{m'=1}^{N} G[m'] \exp\left[i2\pi (m'-1)(N'-1)/N\right]$$

where $N' = 1, 2, 3, \ldots N$

If we just implemented the sums above in code, they would be very slow. Due to the prevalence of Fourier transforms in many fields of science and engineering, much effort has been

put into algorithms which compute the DFT much faster. These types of algorithms are called Fast Fourier Transforms (FFTs). The fastest algorithms occur when N is a power of 2. Other values of N can be used, but tend to be a little slower. Even values of N help the speed as well.

2D DFTs

The concepts above easily generalize to 2D. The main issue to remember is the scaling factors are now scaled to account for each dimension. The proper scaling looks like

```
G = fftshift(fft2(fftshift(g))) * delta^2;
```

Screen clipping taken: 11/1/2019 4:36 PM

where again delta = X_s in our notation. Similarly, the inverse transform is

```
g = ifftshift(ifft2(ifftshift(G))) * (N * delta_f)^2;
```

Screen clipping taken: 11/1/2019 4:38 PM

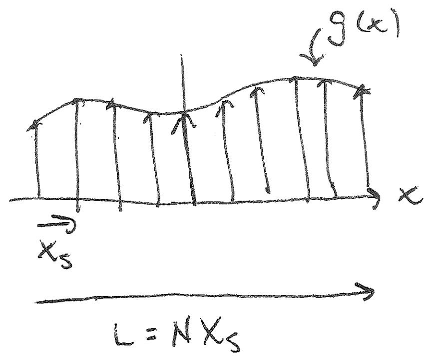
with delta_f = $\frac{1}{N X_s}$ in our notation.

2D CONVOLUTIONS

```
Fa = fft(fa);
Fb = fft(fb);
FO = Fa.* Fb;
fo = ifft(FO) * Xs;
f = fftshift(fo);
```

Convolution uses

Effects of Moving from Continuous to Discrete Transforms



$$g(x) \overset{\mathcal{F}\{\}}{\Rightarrow} G(\xi) \quad \text{CONTINUOUS}$$

$$g(x)\frac{1}{X_s}\text{comb}\left(\frac{x}{X_s}\right) \overset{\mathcal{F}\{\}}{\rightarrow} G(\xi) * \text{comb}(X_s\xi) \quad \text{SAMPLED SPATIAL DOMAIN}$$

$$g(x)\frac{1}{X_s}\text{comb}\left(\frac{x}{X_s}\right)\text{rect}\left(\frac{x}{L}\right) \overset{\mathcal{F}\{\}}{\rightarrow} G(\xi) * \text{comb}(X_s\xi) * L\,\text{sinc}(L\xi) \quad \text{FINITE WIDTH SPATIAL DOMAIN}$$

$$\underbrace{\left[g(x)\frac{1}{X_s}\text{comb}\left(\frac{x}{X_s}\right)\text{rect}\left(\frac{x}{L}\right)\right] * \frac{1}{L}\text{comb}\left(\frac{x}{L}\right)}_{=\,\hat{g}(x)} \overset{\mathcal{F}^{-1}\{\}}{\leftarrow} \underbrace{\left[G(\xi) * \text{comb}(X_s\xi) * L\,\text{sinc}(L\xi)\right]\text{comb}(L\xi)}_{=\,\hat{G}(\xi)} \quad \text{SAMPLED FREQUENCY DOMAIN}$$

($L = NX_s$)

EFFECT	LEADS TO
SPATIAL DOMAIN SAMPLING	ALIASING IN SPATIAL FREQUENCY DOMAIN
FINITE SPATIAL WIDTH	BLURRING BY SINC FUNCTION
FREQUENCY DOMAIN SAMPLING	REPLICATION IN SPATIAL DOMAIN

We want $\hat{g}(x)$ to be as close as possible to $g(x)$. Similarly, we want $\tilde{G}_s(\xi)$ to be as close as possible to $G(\xi)$.

What happens if we increase L?

→ $\text{sinc}(L\xi)$ becomes narrower. There are two ways to increase L. We can increase N or we can increase X_s. Recall the separation between spectra in $\hat{G}(\xi)$ is $\frac{1}{X_s}$, so increasing X_s will reduce the blurring associated with the sinc function, but increases the aliasing

What happens if we increase N, but keep L fixed?

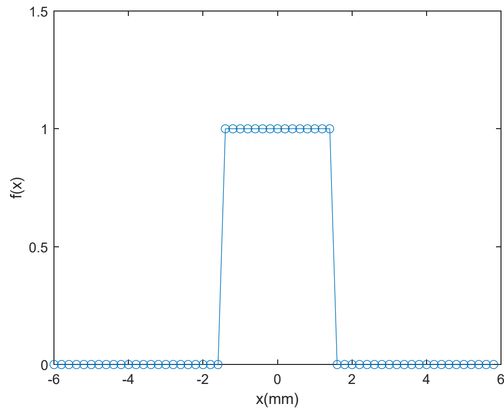
To keep L fixed while N increases, X_s must decrease. This will reduce aliasing, but leave the blurring from the sinc function unchanged.

To help reduce the issues with DFT, in general you want to increase N and decrease X_s. This leads to increased computation time and memory requirements

DFT Examples

```
Editor - C:\Users\jschw\Dropbox\Class\OPTI 51
DFTDemos.m    rect.m
1    function[out]=rect(x)
2    % rectangle function
3 -  out=abs(x)<=1/2;
4 -  end
```

```
Editor - C:\Users\jschw\Dropbox\Class\OPTI 512 Linear Systems, Fourier Transforms
DFTDemos.m    rect.m
1 -   b=3.0; %rectangle width (mm)
2 -   L=12.0; %vector side length (mm)
3 -   N=60; %number of samples
4 -   Xs=L/N; %sample interval (m)
5
6 -   x=-L/2:Xs:L/2-Xs; %coordinate vector
7 -   f=rect(x/b); %rect function at values x
8 -   xi=-1/(2*Xs):1/(N*Xs):1/(2*Xs)-1/(N*Xs);
9 -   F=fft(f);
10
11 -  figure(1)
12 -  plot(x,f,'-o'); %plot f(x) = rect(x/b)
13 -  axis([-6 6 0 1.5]);
14 -  xlabel('x(mm)');
15 -  ylabel('f(x)');
```



Continuous Functions

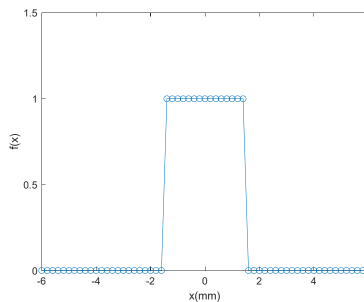
$$f(x) = rect\left(\frac{x}{3}\right)$$

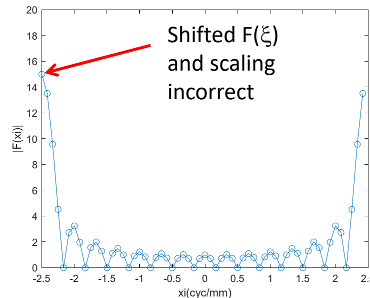
$$F(\xi) = 3sinc(3\xi)$$

DFT Examples

```
Editor - C:\Users\jschw\Dropbox\Class\OPTI 512 Linear Systems, Fourier Transform
DFTDemos.m*    rect.m
1 -   b=3.0; %rectangle width (mm)
2 -   L=12.0; %vector side length (mm)
3 -   N=60; %number of samples
4 -   Xs=L/N; %sample interval (m)
5
6 -   x=-L/2:Xs:L/2-Xs; %coordinate vector
7 -   f=rect(x/b); %rect function at values x
8 -   xi=-1/(2*Xs):1/(N*Xs):1/(2*Xs)-1/(N*Xs);
9 -   F=fft(f);
10
11 -  figure(1)
12 -  plot(x,f,'-o'); %plot f(x) = rect(x/b)
13 -  axis([-6 6 0 1.5]);
14 -  xlabel('x(mm)');
15 -  ylabel('f(x)');
16
17 -  figure(2)
18 -  plot(xi,abs(F),'-o'); %plot F(xi) = b sinc(b*xi)
19 -  axis([-2.5 2.5 0 20]);
20 -  xlabel('xi(cyc/mm)');
21 -  ylabel('F(xi)');
```

Unshifted f(x)





Shifted F(ξ) and scaling incorrect

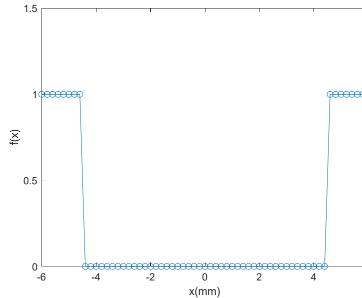
DFT Examples

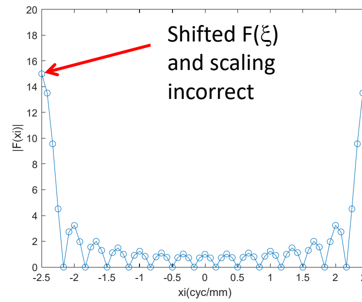
Editor - C:\Users\jschw\Dropbox\Class\OPTI 512 Linear Systems, Fourier Transforms\No

DFTDemos.m | rect.m

```
1 -   b=3.0; %rectangle width (mm)
2 -   L=12.0; %vector side length (mm)
3 -   N=60; %number of samples
4 -   Xs=L/N; %sample interval (m)
5
6 -   x=-L/2:Xs:L/2-Xs; %coordinate vector
7 -   f=fftshift(rect(x/b)); %rect function at values x
8 -   xi=-1/(2*Xs):1/(N*Xs):1/(2*Xs)-1/(N*Xs);
9 -   F=fft(f);
10
11 -  figure(1)
12 -  plot(x,f,'-o'); %plot f(x) = rect(x/b)
13 -  axis([-6 6 0 1.5]);
14 -  xlabel('x(mm)');
15 -  ylabel('f(x)');
16
17 -  figure(2)
18 -  plot(xi,abs(F),'-o'); %plot F(xi) = b sinc(b*xi)
19 -  axis([-2.5 2.5 0 20]);
20 -  xlabel('xi(cyc/mm)');
21 -  ylabel('|F(xi)|');
```

Shifted f(x)





Shifted F(ξ) and scaling incorrect

DFT Examples

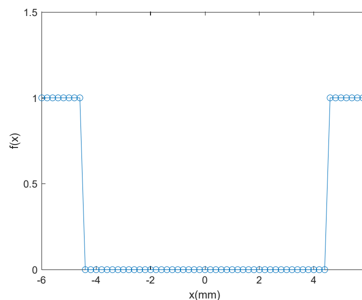
Editor - C:\Users\jschw\Dropbox\Class\OPTI 512 Linear Systems, Fourier Transforms\Notes

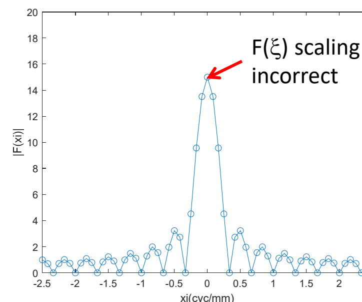
DFTDemos.m | rect.m

```
1 -   b=3.0; %rectangle width (mm)
2 -   L=12.0; %vector side length (mm)
3 -   N=60; %number of samples
4 -   Xs=L/N; %sample interval (m)
5
6 -   x=-L/2:Xs:L/2-Xs; %coordinate vector
7 -   f=fftshift(rect(x/b)); %rect function at values x
8 -   xi=-1/(2*Xs):1/(N*Xs):1/(2*Xs)-1/(N*Xs);
9 -   F=fftshift(fft(f));
10
11 -  figure(1)
12 -  plot(x,f,'-o'); %plot f(x) = rect(x/b)
13 -  axis([-6 6 0 1.5]);
14 -  xlabel('x(mm)');
15 -  ylabel('f(x)');
16
17 -  figure(2)
18 -  plot(xi,abs(F),'-o'); %plot F(xi) = b sinc(b*xi)
19 -  axis([-2.5 2.5 0 20]);
20 -  xlabel('xi(cyc/mm)');
21 -  ylabel('|F(xi)|');
```

Shifted f(x)

Shifted F(ξ)





F(ξ) scaling incorrect

DFT Examples

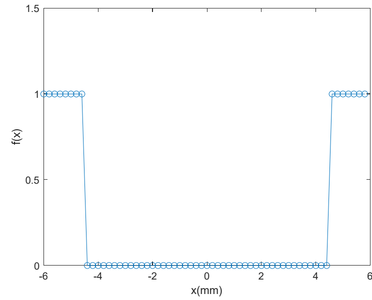
Editor - C:\Users\jschw\Dropbox\Class\OPTI 512 Linear Systems, Fourier Transforms\No

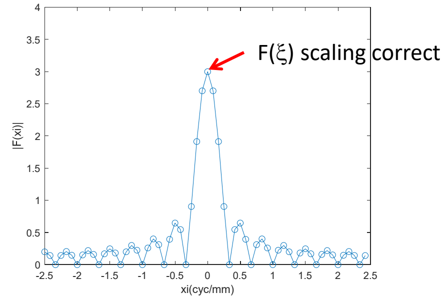
DFTDemos.m | rect.m

```
b=3.0; %rectangle width (mm)
L=12.0; %vector side length (mm)
N=60; %number of samples
Xs=L/N; %sample interval (m)

x=-L/2:Xs:L/2-Xs; %coordinate vector
f=fftshift(rect(x/b)); %rect function at values x
xi=-1/(2*Xs):1/(N*Xs):1/(2*Xs)-1/(N*Xs);
F=Xs*fftshift(fft(f));

figure(1)
plot(x,f,'-o'); %plot f(x) = rect(x/b)
axis([-6 6 0 1.5]);
xlabel('x(mm)');
ylabel('f(x)');

figure(2)
plot(xi,abs(F),'-o'); %plot F(xi) = b sinc(b*xi)
axis([-2.5 2.5 0 4]);
xlabel('xi(cyc/mm)');
ylabel('|F(xi)|');
```

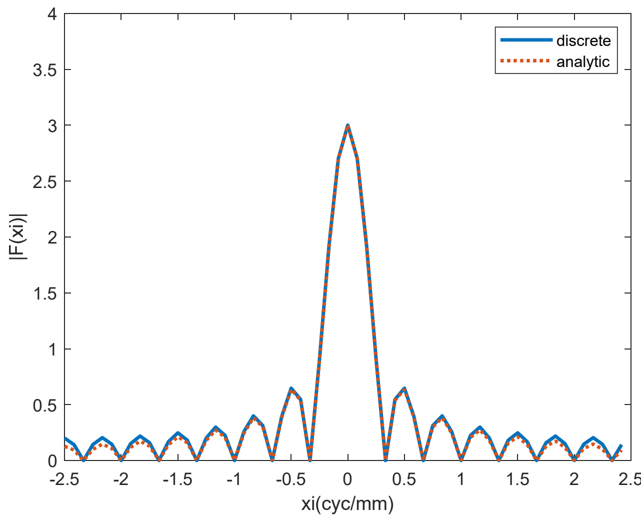
Shifted f(x)

Scaled and Shifted F(ξ)



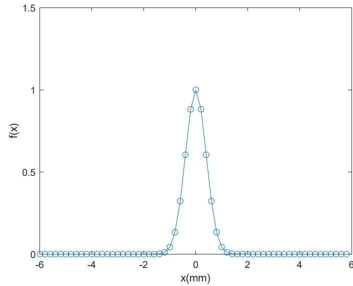


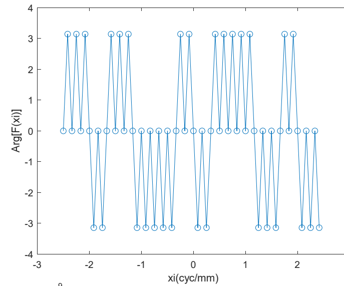
DFT Examples

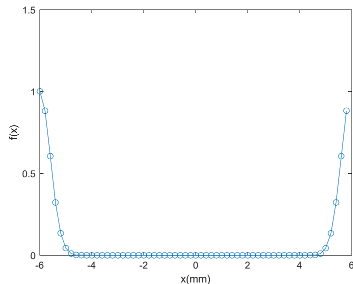


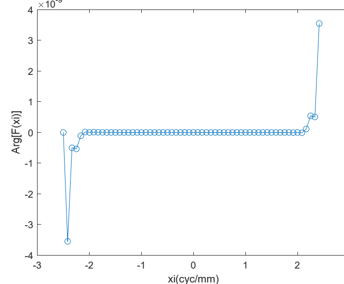
Comparing the discrete and continuous answers shows some small discrepancies near the edges.

Shifted Object and Phase of Transform





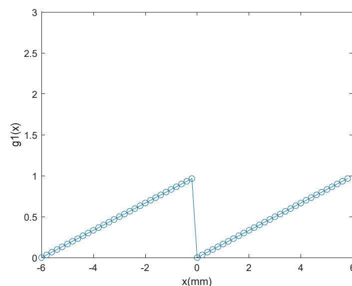


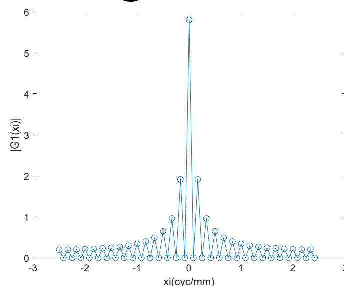


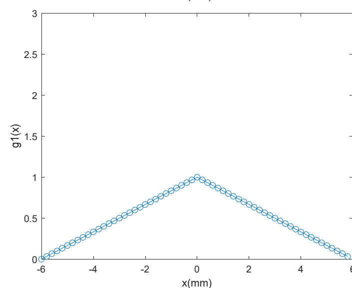
At first glance, the effect of the shift on the object might not be obvious. Here, when the object is centered.

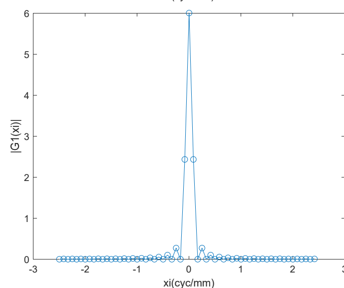
The modulus looks correct for both centered and decentered objects, but the phase only looks correct when the object is shifted as well.

Discontinuities at Edges



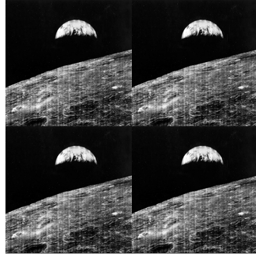
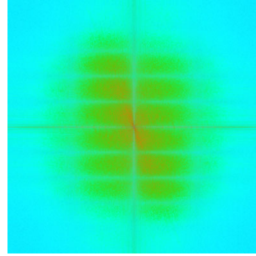
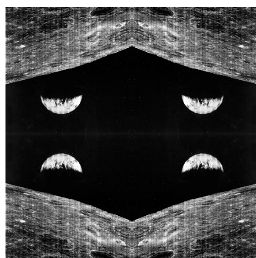
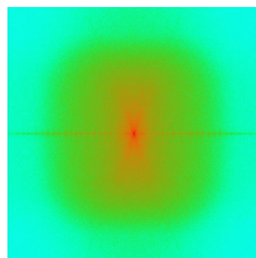






Small values for high frequencies

Discontinuities at Edges

Since the DFT inherently replicates the object, any discontinuities across the boundaries of this replication causes high frequency values in the transform which leads to aliasing.

This issue can be reduced by creating an object that is mirrored about its horizontal and vertical edges. This leads to a smoother transition at the boundaries, lower frequency values and less aliasing.

Convolution

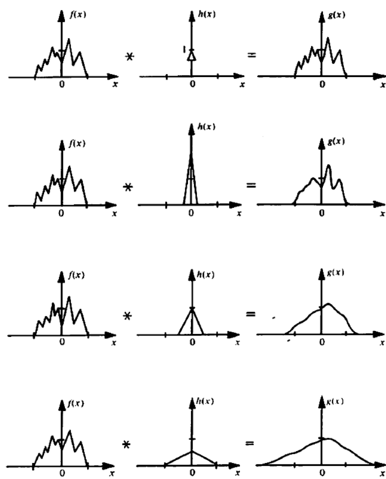


Figure 6-5 Smoothing effects of convolution.

Convolution tends to smooth out high frequency and lead to a resulting function that has a "width" equal to the sum of the widths of the two functions being convolved.

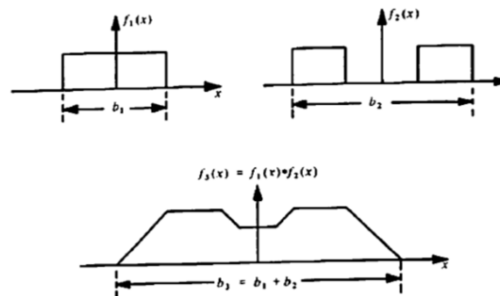
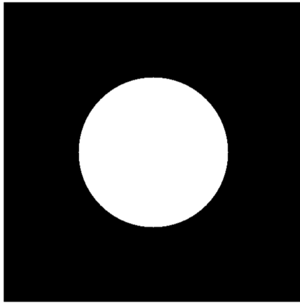
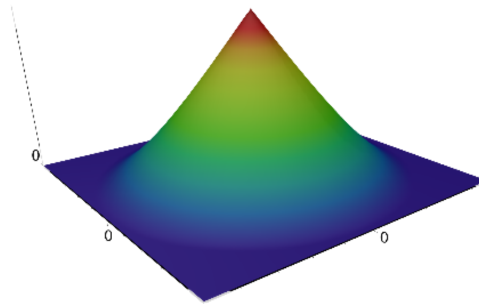


Figure 6-6 Convolution of two functions with compact support.

Convolution & Correlation





In general, when calculating convolution & correlation, the size of the array should be at least the size of the sum of the two functions being convolved (correlated). Here a cyl() function is shown along with its autocorrelation. The array needs to be twice the diameter of the cylinder to avoid aliasing. This is sometimes referred to as “padding” the array.