Problem 2

(a) Forward model (continuous operator):

(a) Forward models: The forward model can be described in continuous and the discrete forms

- Continuous forward model:

The question states that the system is LSI (input-output) and satisfies the following convolution formula:

$$g_{out}(x,y) = g_{in}(x,y;z) * h(x,y;z) \quad \text{—(1)}$$

where $*$ is the ~~cancel~~ 2D convolution and

$$h(x,y;z) = \frac{1}{j\lambda z} e^{jk\frac{x^2+y^2}{2z}}.$$

- Let $\mathcal{A}$ denote the operator of the system, then the continuous operator form (Lectures 11-13, slide 22) is given by:

$$(\mathcal{A}g_{in})(x,y) = h(x,y;z) * g_{in}(x,y;z)$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x-x', y-y'; z)\, g_{in}(x',y';z)\, dx'dy'$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{j\lambda z} e^{jk\left(\frac{(x-x')^2+(y-y')^2}{2z}\right)} g_{in}(x',y';z)\, dx'dy'$$

- Alternatively in Spectral Form:

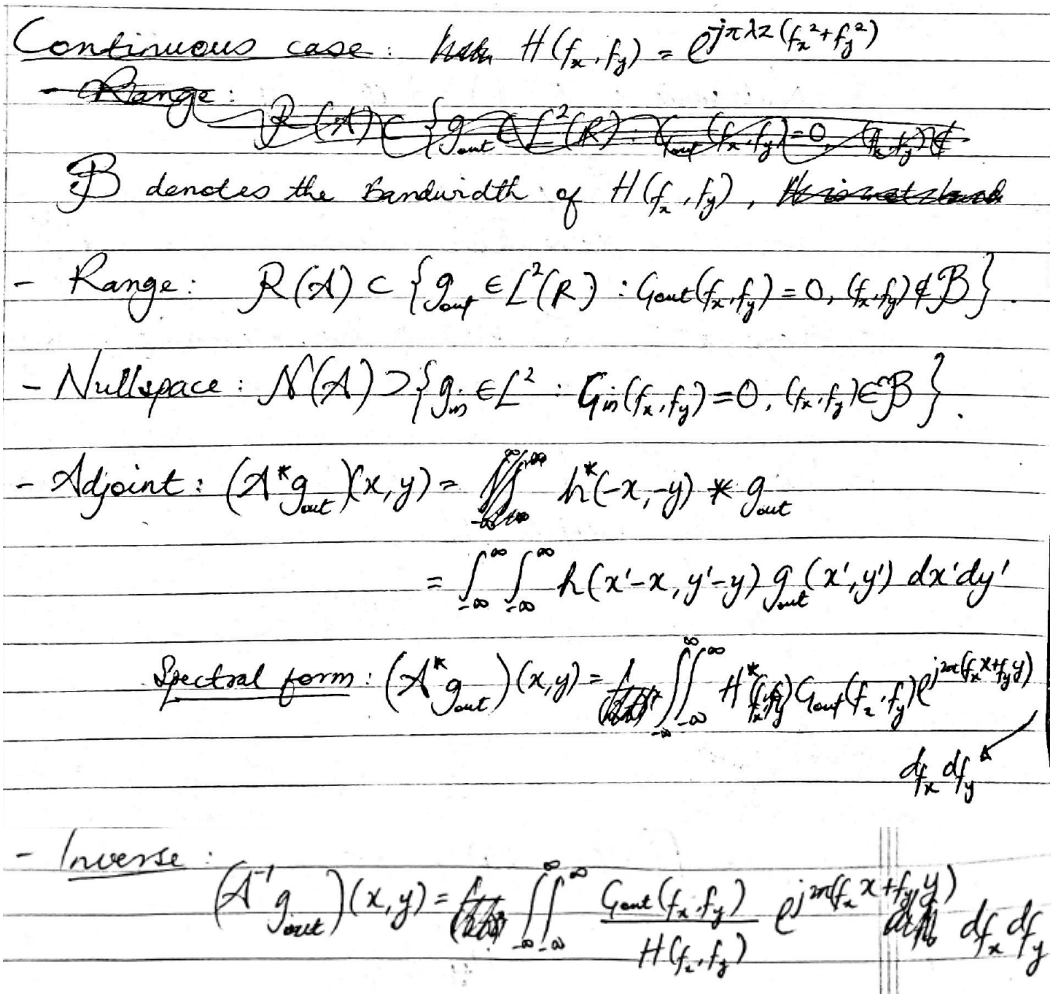
$$(\mathcal{A}g_{in})(x,y) = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(\omega_x', \omega_y'; z)\, G_{in}(\omega_x', \omega_y'; z) e^{j(\omega_x' x + \omega_y' y)}\, d\omega_x' d\omega_y'$$

letting $\omega_x' = 2\pi f_x$ and $\omega_y' = 2\pi f_y \Rightarrow d\omega_x' = 2\pi df_x$ and $d\omega_y' = 2\pi f_y$.

Thus, $(\mathcal{A}g_{in})(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(f_x, f_y; z)\, G_{in}(f_x, f_y; z) e^{j2\pi(f_x x + f_y y)}\, df_x\, df_y$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-j\pi\lambda z(f_x^2+f_y^2)} e^{j2\pi(f_x x + f_y y)} G_{in}(f_x, f_y; z)\, df_x\, df_y$$

(b) Operator properties (properties of the continuous operator):



(c) Discrete forward model (discrete operator):

- The Discrete Forward model: To get the discrete forward model, or the Matrix form of the system we need to discretize the object space, and also sample the system output (i.e. pixel passer function).

(i) Sampling: Assume square pixels of size Δ, with pixel passer function (in 2D):

$$\rho_m(x,y) = \text{rect}\left(\frac{x-m_1\Delta}{\Delta}\right)\text{rect}\left(\frac{y-m_2\Delta}{\Delta}\right)$$

$$= \rho_0(x-m_1\Delta,\ y-m_2\Delta)$$

where $\rho_0(x,y) = \text{rect}\left(\frac{x}{\Delta}\right)\text{rect}\left(\frac{y}{\Delta}\right)$ is the 2D box car function with width Δ. Note that m_1 and m_2 are column and row indexes for the pixel location, $m=(m_1, m_2)$.

(ii) Discretization of object function: assuming the standard, or canonical basis (Lecture 14-15), this means the basis $\psi_n(x,y)$ are a set of 2D box-car functions of width δ in each dimension, thus:

$$\psi_n(x,y) = \text{rect}\left(\frac{x-n_1\delta}{\delta}\right)\text{rect}\left(\frac{y-n_2\delta}{\delta}\right) = \psi_0(x-n_1\delta)\psi_0(y-n_2\delta)$$

where $\psi_0(x,y) = \text{rect}\left(\frac{x}{\delta}\right)\text{rect}\left(\frac{y}{\delta}\right)$.

Again (n_1, n_2) denote the position of the basis ψ_n.

Then the fully discrete model is given by the matrix A, with (m,n) entries:

$$A_{m,n} = \langle \psi_n, \rho_m \rangle$$

{ Remember that the angled brackets denote the inner product.

$$= \iiint \psi_0(x'-n_1\delta, y'-n_2\delta)\, h(x-x', y-y')\, \rho_0(x-m\Delta, y-m_2\Delta)\, dx'dx\, dy'dy$$

$$= \left(\psi_0 * h * \rho_0\right)(m_1\Delta - n_1\delta,\ m_2\Delta - n_2\delta)$$

$$A_{mn} = \iint \underbrace{H(f_x, f_y)\, P_o(f_x, f_y)\, \Psi_o(f_x, f_y)}_{H_{sys}(f_x, f_y)} e^{j2\pi[(m_1\Delta - n_1\delta)f_x + (m_2\Delta - n_2\delta)f_y]} \, df_x \, df_y$$

Where $H(f_x, f_y) \xleftrightarrow{FT} h(x,y)$, $P_o(f_x, f_y) \xleftrightarrow{FT} p_o(x,y)$ and $\Psi_o(f_x, f_y) \xleftrightarrow{FT} \psi_o(x,y)$.

Note that the last integral above follows from the convolution property of the Fourier transform. (We saw this in-class.)
H_{sys} is the system transfer function.

(d) Discrete operator properties

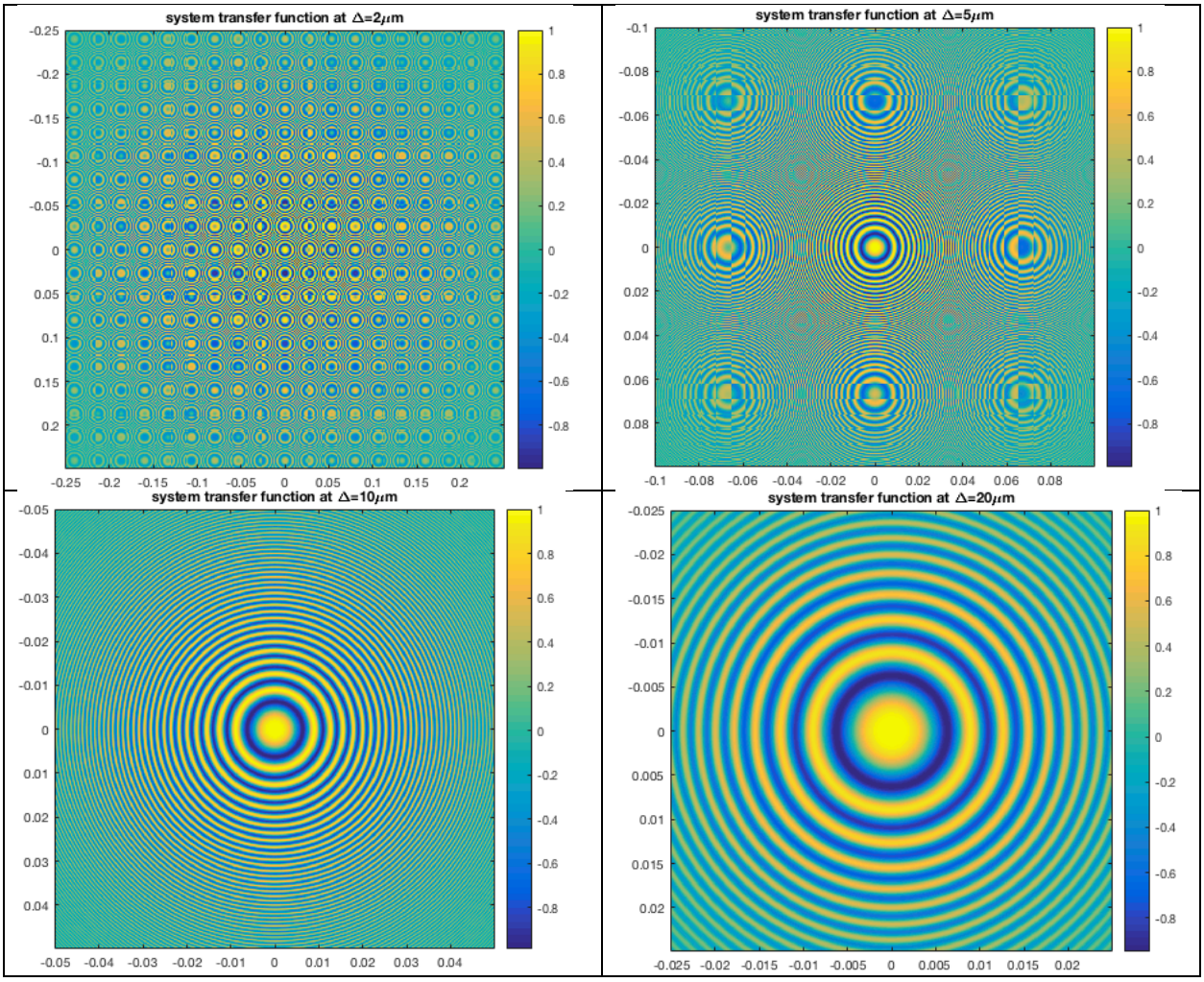
- Discrete Case: (The same as defined for matrices)
  - Range: $\text{Range}(A) = \{ g_{out} : Ag_{in} = g_{out} \}$
    The column space of A or the eigenvectors of A corresponding to non-zero eigenvalues.
  - Nullspace: $N(A) = \{ g_{in} : Ag_{in} = 0 \}$
    The eigenvectors of A corresponding to the zero eigenvalues.
  - Adjoint: A^* is just the Hermitian transpose of the matrix A with entries A_{mn}
  - Inverse: A^{-1} is the inverse of the matrix A with entries A_{mn}.

(e) MATLAB code – note that because of the size of the 4D tensor (indexed by (m_1, m_2, n_1, n_2) each index running from 0 to 999), it is far more efficient to implement this operator in spectral form and exploiting the convolution property of the FT. Precisely, each of the four dimensions of the tensor has 1000 elements, making a 4D array with $1000^4 = 10^{12}$ elements, MATLAB uses double precision floating-point representation for each element (i.e., 8 bytes per element), thus the array is roughly of size 8×10^{12} bytes or 8 terabytes. You will need 8 TB of RAM to be able to initialize the array, before even doing any computation.
Thanks to the FFT we can circumvent this issue. When the object discretization and the sampling are vectorized (through a lexicographic relabelling) we obtain a **block-Toeplitz matrix**, as such a single column of that matrix is sufficient to describe the entire system. This is analogous to the 1D example we completed in-class (end of Lecture 15 and 16), wherein we ended up with a 2D matrix. Here however, we are in 2D, thus one obtains a 4D Tensor, which can be turned into a matrix using a lexicographic ordering. The code to achieve this follows:

```
%% define Fourier transform operator
%% notice the use of fftshift & ifftshift
F = @(g) fftshift(fft(ifftshift(g)));
Ft = @(G) fftshift(ifft(ifftshift(G)));
%% system parameters
lambda = 0.5; % wavelength of light, in micrometers
N = 1000; % number of pixels
k = 2*pi/lambda;
%% 2D holographic imaging: problem (f)(i)
sampling size
Delta = [2,5,10,20];
% z-distance
z = 50e3;
for m = 1:length(Delta)
set up spatial frequency coordinates
du = 1/N/Delta(m);
% linear coordinates
u = [-N/2:N/2-1]*du;
% setup 2D coordinates
[uu,vv] = meshgrid(u);
%% Calculate the pixel transfer function P
assuming square pixels of size Delta the pixel transfer function is the Fourier transform of the pixel sampling function
P = sinc(Delta(m)*uu).*sinc(Delta(m)*vv);
% the discretization transfer function is the Fourier transform of the
% discretization function
Q = P;
calculate the optical transfer function
H = exp(-1i*pi*lambda*z*(uu.^2+vv.^2));
calculate the system transfer function
```

```
H_sys = H.*P.*Q;
visualize the results
figure(m);
imagesc(u,u,real(H_sys)); axis image; colorbar;
title(['system transfer function at \Delta=',num2str(Delta(m)),'\mum'])
```

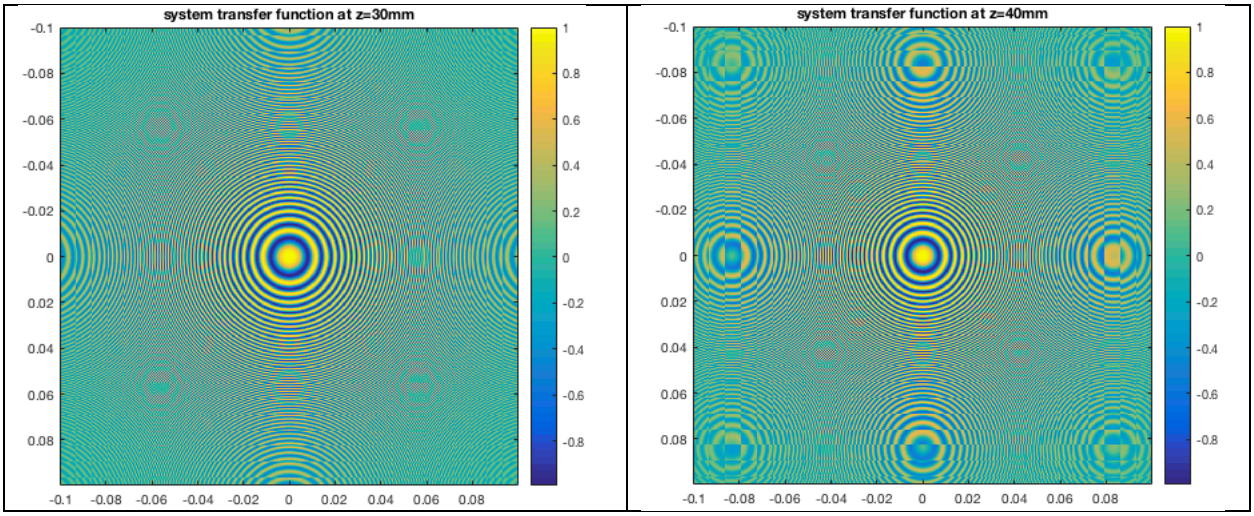
(f) (i) – The system transfer function

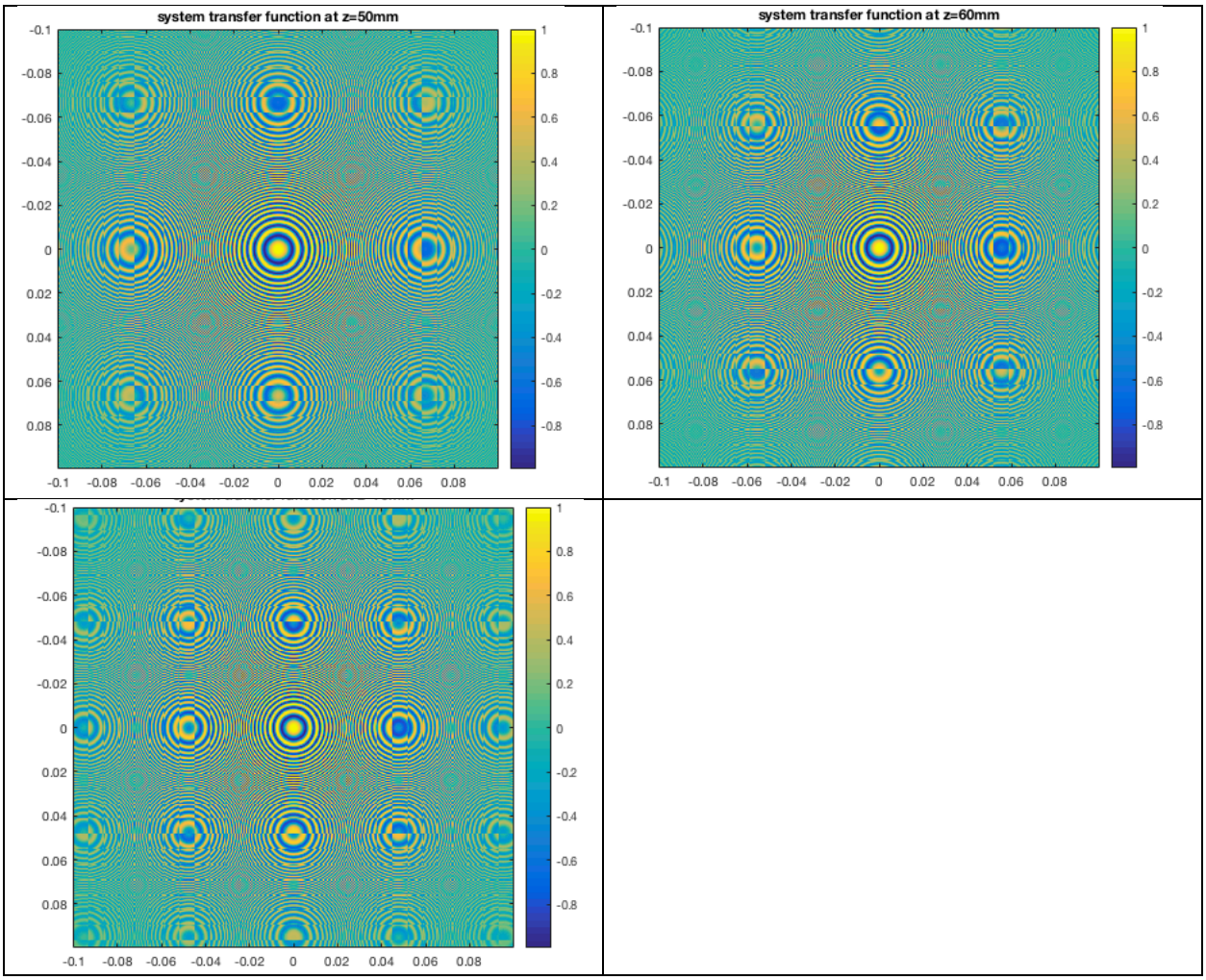


(ii) MATLAB Code

```
%% sampling size
Delta = 5;
%% set up spatial frequency coordinates
```

```
du = 1/N/Delta;
% linear coordinates
u = [-N/2:N/2-1]*du;
% setup 2D coordinates
[uu,vv] = meshgrid(u);
%% Calculate the pixel transfer function P
assuming square pixels of size Delta the pixel transfer function is the Fourier transform of the pixel sampling function
P = sinc(Delta*uu).*sinc(Delta*vv);
% the discretization transfer function is the Fourier transform of the
% discretization function
Q = P;
% z-distance
z = [30:10:70]*1e3;
for m = 1:length(z)
%% calculate the optical transfer function
H = exp(-1i*pi*lambda*z(m)*(uu.^2+vv.^2));
%% calculate the system transfer function
H_sys = H.*P.*Q;
%% visualize the results
figure;
imagesc(u,u,real(H_sys)); axis image; colorbar;
title(['system transfer function at z=',num2str(z(m)/1e3),'mm']
end
```





(iii)

```
Delta = 5;
% z-distance
z = 50e3;
% discretization size
delta = [1:2:10];
```