The Discrete Fourier Transform

Dr. Daniel B. Rowe
Professor of Computational Statistics
Department of Mathematical and Statistical Sciences
Marquette University

Outline

Introduction

1D Discrete Fourier Transform

2D Discrete Fourier Transform

FMRI Application

Discussion

Homework

Introduction

Fourier analysis and the Fourier transform are useful techniques.

The Fourier transform is a way of representing a function as a linear combination of cosines and sines similar to how a Taylor series represents a function as a linear combination of polynomials.

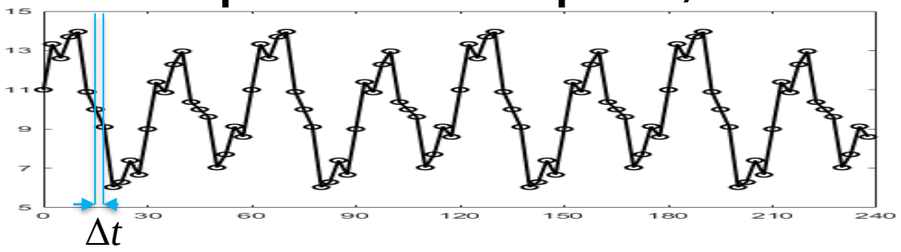
Because of the repeating nature of cosines and sines, the Fourier transform is ideal for periodic functions.

There is some great math here but we will be using the discrete Fourier transform in an applied setting. There will be some math.

1D Discrete Fourier Transform

$$e^{-i\theta} = \cos(\theta) - i\sin(\theta)$$

If we have a (periodic) time series $y(t)$ sampled at n time points Δt apart, the 1D forward discrete Fourier transform is



$$f(q\Delta\nu) = \sum_{p=-n/2}^{n/2-1} y(p\Delta t) e^{-\frac{i2\pi}{n}pq} \qquad \text{for } q = 1,\ldots,n.$$

Zero frequency in center.

The difference in temporal frequency $\Delta\nu$ is

$$\Delta\nu = \frac{1}{n\Delta t} \quad .$$

Note that $f(q\Delta\nu)$ is a complex-valued quantity, $i = \sqrt{-1}$.
The result is a vector of amplitude coefficients for the various frequencies.

Fourier

1D Discrete Fourier Transform

We can recover $y(p\Delta t)$ by taking the 1D inverse discrete Fourier transform.

$$y(p\Delta t) = \sum_{q=-n/2}^{n/2-1} f(q\Delta\nu) e^{\frac{i2\pi}{n}pq} \qquad \text{for } p = 1,\ldots,n.$$

We can only resolve frequencies up to

$$\nu_{\max} = \frac{n}{2}\Delta\nu$$

and again the frequency resolution is

$$\Delta\nu = \frac{1}{n\Delta t}.$$

Larger n resolved more frequencies and smaller Δt yields higher ν_{max}.

1D Discrete Fourier Transform

This time series y and the resolved frequencies f can be written as

$$\underset{n\times 1}{y} = \begin{bmatrix} y(1\Delta t) \\ y(2\Delta t) \\ \vdots \\ y((n-1)\Delta t) \\ y(n\Delta t) \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} \quad \text{and} \quad \underset{n\times 1}{f} = \begin{bmatrix} f(-n\Delta\nu/2) \\ f(-(n-1)\Delta\nu/2) \\ \vdots \\ f((n/2-2)\Delta\nu/2) \\ f((n/2-1)\Delta\nu/2) \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

We now need to define a Fourier transform matrix Ω to convert y to f.

1D Discrete Fourier Transform

The forward discrete Fourier matrix Ω is

$$\underset{n\times n}{\Omega} = \begin{bmatrix} W^{(-\frac{n}{2})(-\frac{n}{2})} & W^{(-\frac{n}{2})(-\frac{n}{2}+1)} & \cdots & W^{(-\frac{n}{2})(\frac{n}{2}-1)} \\ W^{(-\frac{n}{2}+1)(-\frac{n}{2})} & W^{(-\frac{n}{2}+1)(-\frac{n}{2}+1)} & & W^{(-\frac{n}{2}+1)(\frac{n}{2}-1)} \\ & & \ddots & \vdots \\ & & & W^{(\frac{n}{2}-2)(\frac{n}{2}-1)} \\ W^{(\frac{n}{2})(-\frac{n}{2})} & & \cdots & W^{(\frac{n}{2}-1)(\frac{n}{2}-1)} \end{bmatrix}$$

where $W = e^{-\frac{i2\pi}{n}} = \cos(\frac{i2\pi}{n}) - i\sin(\frac{i2\pi}{n})$.

1D Discrete Fourier Transform

$$W = e^{-\frac{i2\pi}{n}} = \cos(\tfrac{i2\pi}{n}) - i\sin(\tfrac{i2\pi}{n})$$

The forward discrete Fourier transform is completed by multiplying.

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix} = \begin{bmatrix} W^{(-\frac{n}{2})(-\frac{n}{2})} & W^{(-\frac{n}{2})(-\frac{n}{2}+1)} & \cdots & W^{(-\frac{n}{2})(\frac{n}{2}-1)} \\ W^{(-\frac{n}{2}+1)(-\frac{n}{2})} & W^{(-\frac{n}{2}+1)(-\frac{n}{2}+1)} & & W^{(-\frac{n}{2}+1)(\frac{n}{2}-1)} \\ & & \ddots & \vdots \\ & & & W^{(\frac{n}{2}-2)(\frac{n}{2}-1)} \\ W^{(\frac{n}{2})(-\frac{n}{2})} & & \cdots & W^{(\frac{n}{2}-1)(\frac{n}{2}-1)} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

$$y = y_R + iy_I$$

$$\Omega = \Omega_R + i\Omega_I$$

$$f = f_R + if_I$$

complex-valued

$$\underset{n\times 1}{f} = \underset{n\times n}{\Omega}\ \underset{n\times 1}{y}$$

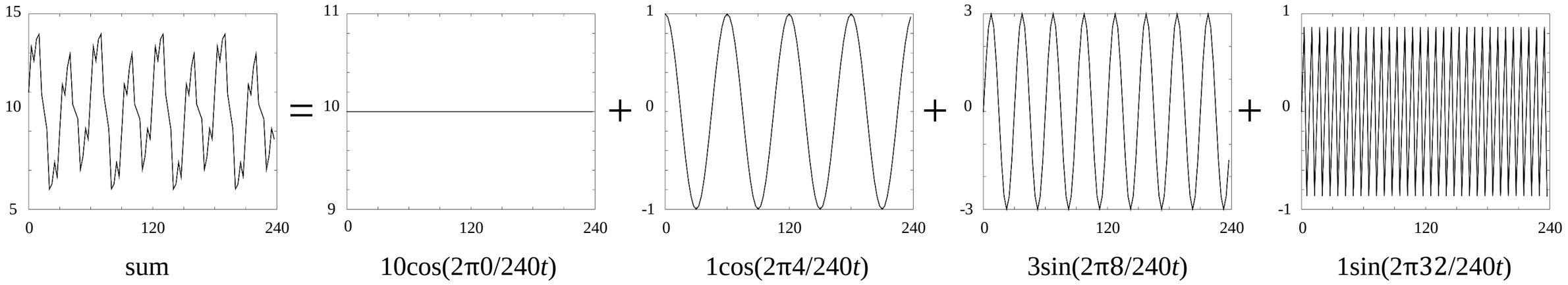
complex-valued | complex-valued | complex-valued

1D Discrete Fourier Transform

Example: Let's sample the continuous time series (1D function)

$$y(t) = 10\cos\left(2\pi\frac{0}{240}t\right) + \cos\left(2\pi\frac{4}{240}t\right) + 3\sin\left(2\pi\frac{8}{240}t\right) + \sin\left(2\pi\frac{32}{240}t\right)$$

at $t=1\Delta t,2\Delta t,3\Delta t,\ldots,n\Delta t$, where $n=96$ and $\Delta t=2.5$s for a total time of 240s.



1D Discrete Fourier Transform

The forward discrete Fourier transform is computed by multiplying.

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix} = \begin{bmatrix} W^{(-\frac{n}{2})(-\frac{n}{2})} & W^{(-\frac{n}{2})(-\frac{n}{2}+1)} & \cdots & W^{(-\frac{n}{2})(\frac{n}{2}-1)} \\ W^{(-\frac{n}{2}+1)(-\frac{n}{2})} & W^{(-\frac{n}{2}+1)(-\frac{n}{2}+1)} & & W^{(-\frac{n}{2}+1)(\frac{n}{2}-1)} \\ & & \ddots & \vdots \\ & & & W^{(\frac{n}{2}-2)(\frac{n}{2}-1)} \\ W^{(\frac{n}{2})(-\frac{n}{2})} & & \cdots & W^{(\frac{n}{2}-1)(\frac{n}{2}-1)} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

$$y = y_R + iy_I$$

$$\Omega = \Omega_R + i\Omega_I$$

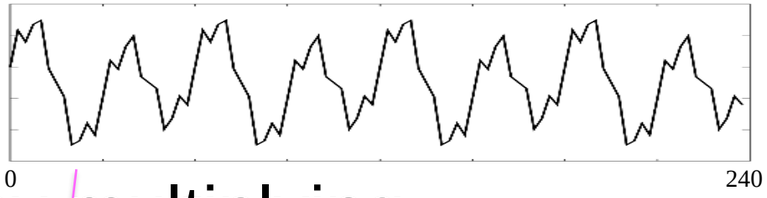
$$f = f_R + if_I$$

complex-valued

$$\underset{n\times 1}{f} = \underset{n\times n}{\Omega} \underset{n\times 1}{y}$$

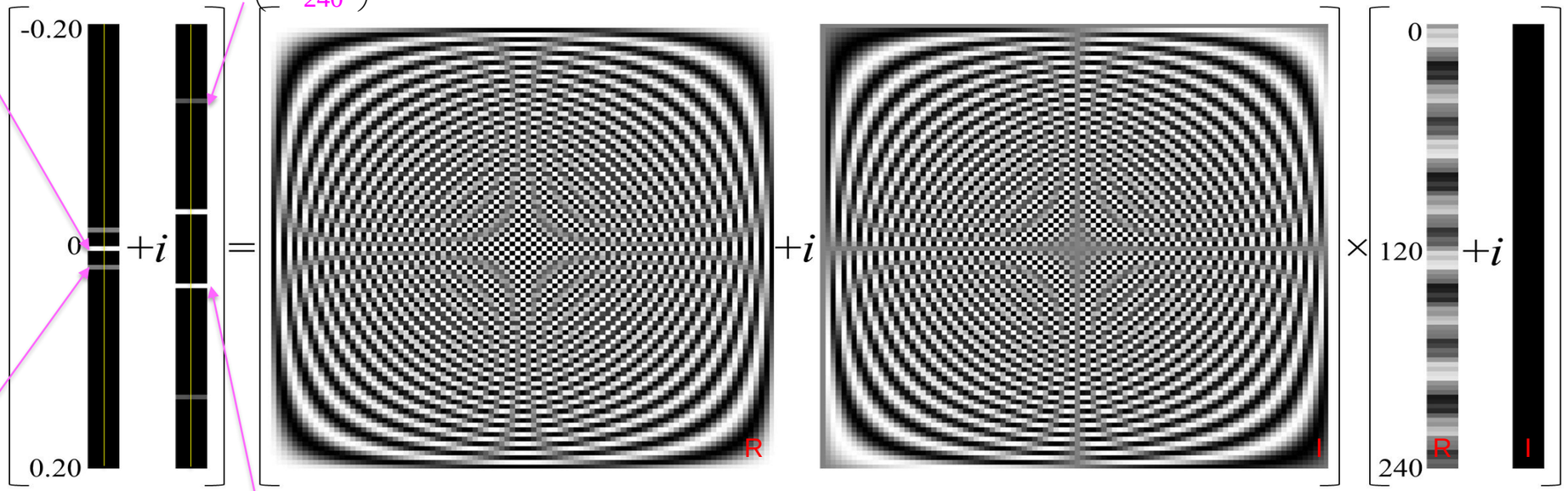
complex-valued | complex-valued | complex-valued

1D Discrete Fourier Transform



The forward discrete Fourier transform is computed by multiplying.

$10\cos\left(2\pi\frac{0}{240}t\right)$ $\sin\left(2\pi\frac{32}{240}t\right)$



$\cos\left(2\pi\frac{4}{240}t\right)$ $3\sin\left(2\pi\frac{8}{240}t\right)$

$$y = y_R + iy_I$$

$$\Omega = \Omega_R + i\Omega_I$$

$$f = f_R + if_I$$

complex-valued

cosines

sines

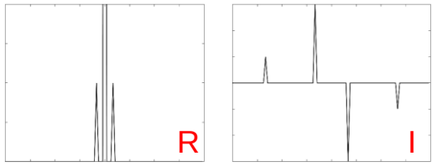
zero imaginary for real-valued time series

$$\underset{n\times 1}{f} = \underset{n\times n}{\Omega} \underset{n\times 1}{y}$$

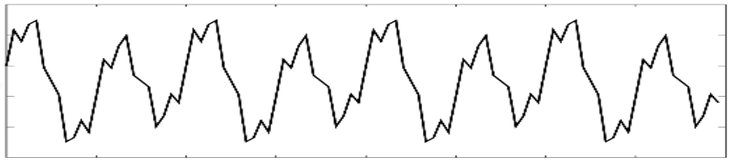
complex-valued

complex-valued

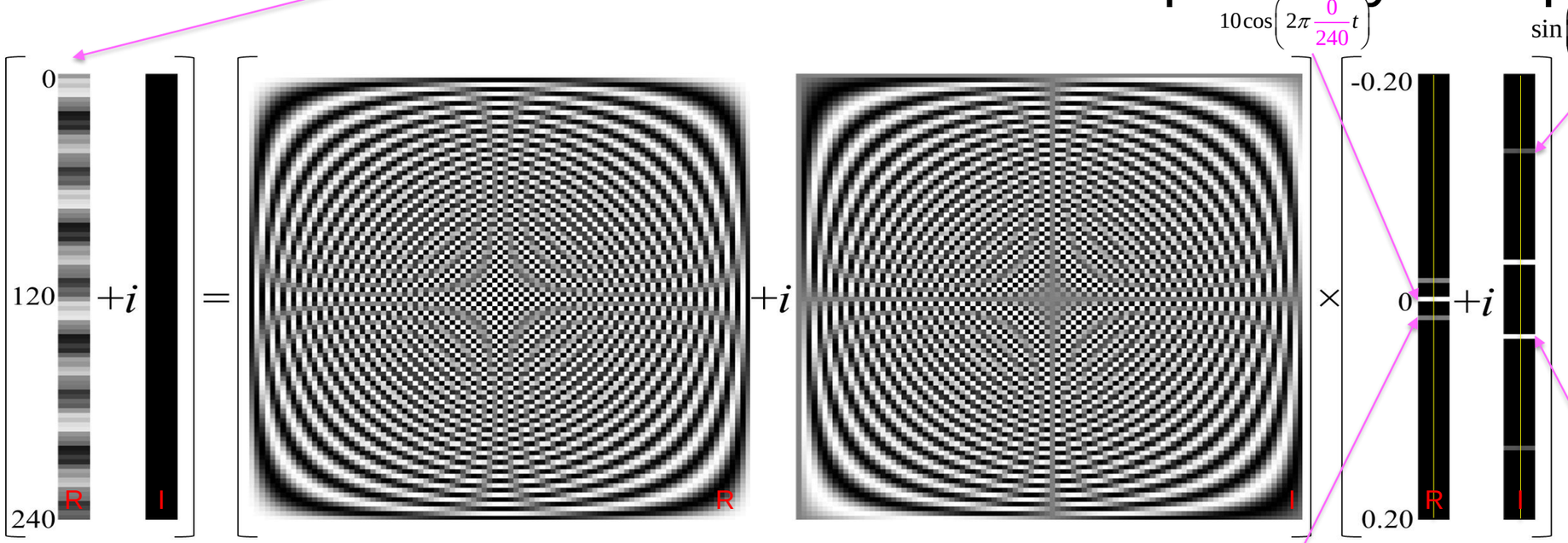
complex-valued



1D Discrete Fourier Transform



The inverse discrete Fourier transform is computed by multiplying.



$$y = y_R + iy_I$$

$$\bar{\Omega} = \bar{\Omega}_R + i\bar{\Omega}_I$$

$$f = f_R + if_I$$

complex-valued

zero for symmetric

cosines

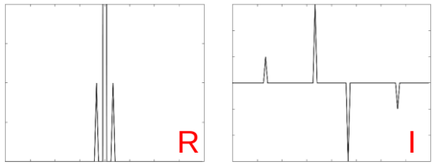
sines

$$\underset{n\times 1}{y} = \underset{n\times n}{\bar{\Omega}} \underset{n\times 1}{f}$$

complex-valued

complex-valued

complex-valued



1D Discrete Fourier Transform

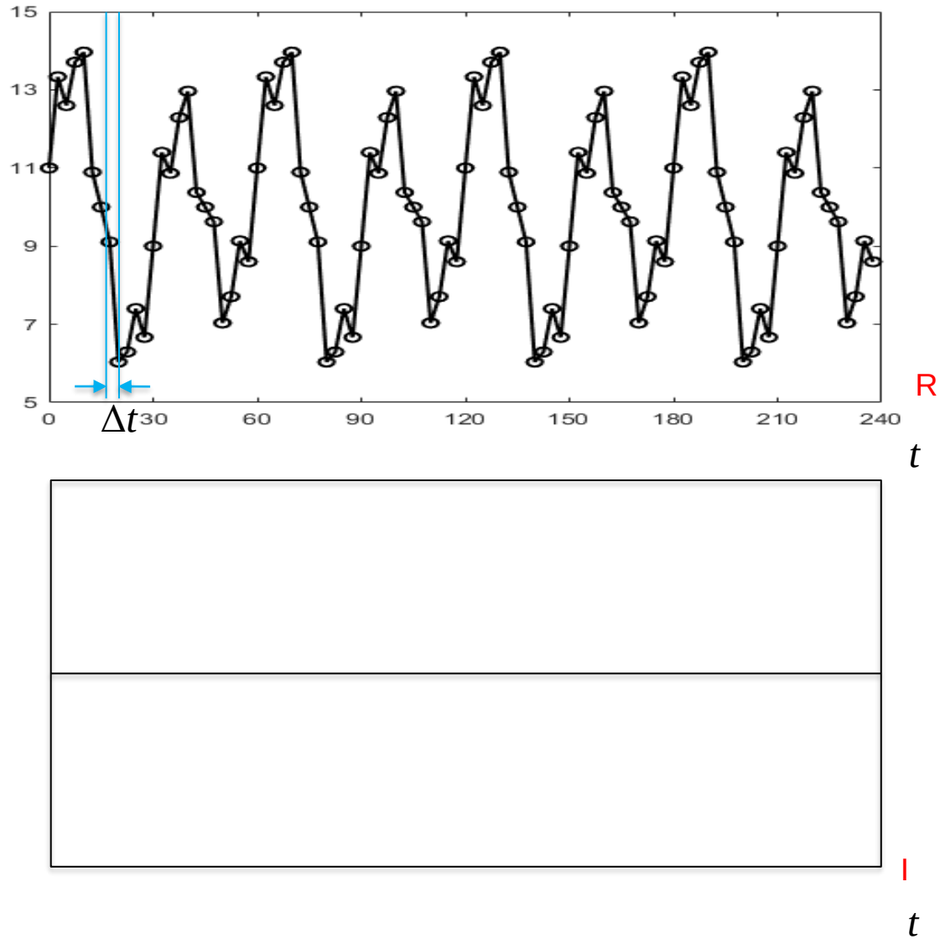
$n=96$

$\Delta t=2.5\text{s}$

$$\Delta\nu=\frac{1}{n\Delta t}=0.0042Hz$$

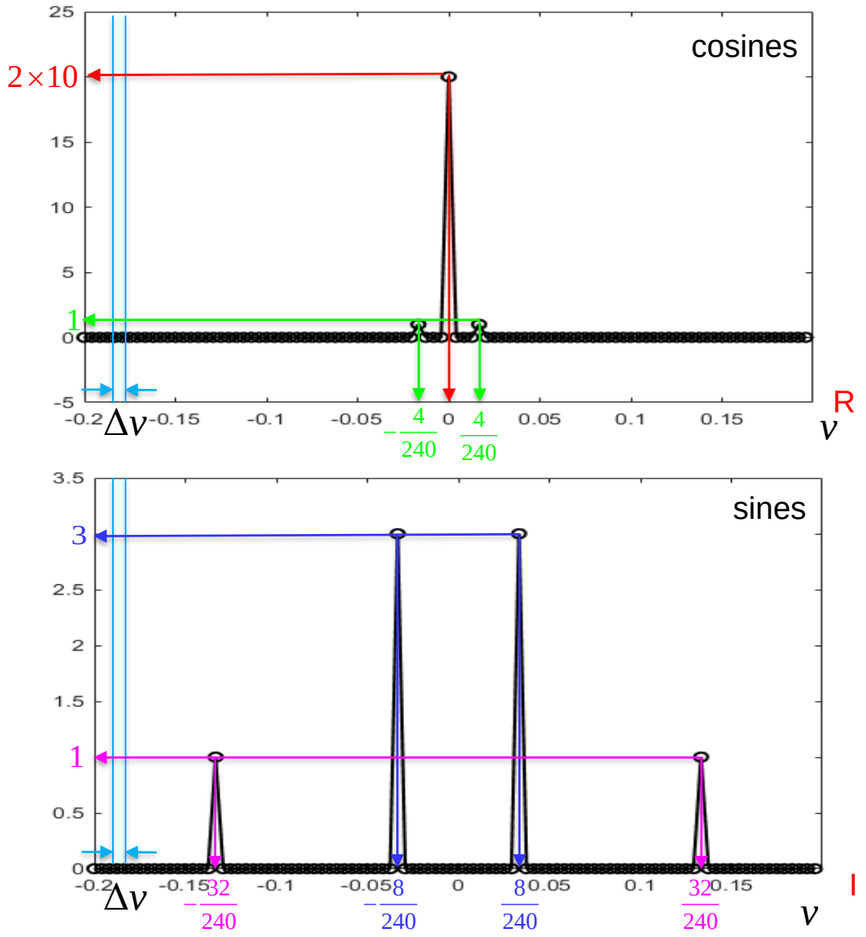
$$\nu_{\max}=\frac{n}{2}\Delta\nu=20Hz$$

Time Series



$\mathcal{F}$

Amplitude Coefficients



$$y(t)=10\cos\left(2\pi\frac{0}{240}t\right)+1\cos\left(2\pi\frac{4}{240}t\right)+3\sin\left(2\pi\frac{8}{240}t\right)+1\sin\left(2\pi\frac{32}{240}t\right)$$

*coefficients divided by $n/2$ for display

1D Discrete Fourier Transform

```
n=96; % number of time points
dt=2.5; %seconds between time points
t=((1:n)'-1)*dt; % sampled time points

A1=10; A2=1; A3=3; A4=1; % frequency amplitudes
nu1=0/(dt*n); nu2=4/(dt*n); % cosine frequencies
nu3=8/(dt*n); nu4=32/(dt*n); % sine frequencies
y1=zeros(n,1); y2=zeros(n,1); y3=zeros(n,1); y4=zeros(n,1);
for j=1:n
    y1(j,1)=A1*cos(2*pi*nu1*(j-1)*dt);
    y2(j,1)=A2*cos(2*pi*nu2*(j-1)*dt);
    y3(j,1)=A3*sin(2*pi*nu3*(j-1)*dt);
    y4(j,1)=A4*sin(2*pi*nu4*(j-1)*dt);
end
y=y1+y2+y3+y4;

figure;
plot(t,y,'k','LineWidth',1.5)
xlim([0,dt*n]), ylim([5,15])
set(gca,'xtick',(0:dt*n/8:dt*n),'FontSize',12,'FontWeight','bold')
set(gca,'ytick',(5:2:15),'FontSize',12,'FontWeight','bold')
```

```
figure;
imagesc(repmat(real(y),[1,n/16]),[5,15])
axis image, colormap(gray), axis off
figure;
imagesc(repmat(abs(imag(y)),[1,n/16]),[0,1])
axis image, colormap(gray), axis off
```

1D Discrete Fourier Transform

```
% DFT by matrix multiplication
dnu=1/(n*dt); % frequency spacing
nu=((1:n)'-n/2-1)*dnu; % resolved frequencies

OmegaC=zeros(n,n);
for k=1:n
   for j=1:n
      OmegaC(k,j)=exp(-1i*2*pi*(k-(n/2+1))*(j-(n/2+1))/n);
   end
end
figure;
imagesc(real(OmegaC),[-1,1]);
axis image,colormap(gray), axis off
figure;
imagesc(imag(OmegaC),[-1,1]);
axis image,colormap(gray), axis off
```

```
figure;
imagesc(repmat(real(f),[1,n/16]),[0,n]) %A1*n
axis image, colormap(gray), axis off
figure;
imagesc(repmat(abs(imag(f)),[1,n/16]),[0,A3*n/2])
axis image, colormap(gray), axis off
```

```
f =(OmegaC*y); % Fourier coefficients
figure;, plot(nu,real(f),'k','LineWidth',1.5)
xlim([-1/dt/2,1/dt/2]),ylim([0,n])
set(gca,'xtick',round(nu(1:n/8:n),2)),set(gca,'ytick',(0:n/4:n))
figure;, plot(nu,abs(imag(f)),'k','LineWidth',1.5)
xlim([-1/dt/2,1/dt/2]),ylim([0,n])
set(gca,'xtick',round(nu(1:n/8:n),2)),set(gca,'ytick',(0:n/4:n))
```

1D Discrete Fourier Transform

```
% IDFT by matrix multiplication
OmegaBarC=zeros(n,n);
for k=1:n
   for j=1:n
      OmegaBarC(k,j)=exp(1i*2*pi*(k-(n/2+1))*(j-(n/2+1))/n);
   end
end
OmegaBarC=OmegaBarC/n;
figure;
imagesc(real(OmegaBarC),[-1/n,1/n]);
axis image,colormap(gray), axis off
figure;
imagesc(imag(OmegaBarC),[-1/n,1/n]);
axis image,colormap(gray), axis off
yhat =(OmegaBarC*f);
figure;
plot(t,real(yhat),'k','LineWidth',1.5)
xlim([0,dt*n]), ylim([5,15])
set(gca,'xtick',(0:dt*n/8:dt*n))
set(gca,'ytick',(5:2:15))
figure;
plot(t,imag(yhat),'k','LineWidth',1.5)
xlim([0,dt*n])
set(gca,'xtick',(0:dt*n/8:dt*n))
```

1D Discrete Fourier Transform

Matlab has built in functions that perform the discrete 1D Fourier transform.

```
ff=fftshift(fft(fftshift(y)));
figure;
plot(nu,real(ff),'k','LineWidth',1.5)
xlim([-1/dt/2,1/dt/2]),ylim([0,n])
set(gca,'xtick',round(nu(1:n/8:n),2))
set(gca,'ytick',(0:n/4:n))
figure;
plot(nu,abs(imag(ff)),'k','LineWidth',1.5)
xlim([-1/dt/2,1/dt/2]),ylim([0,n])
set(gca,'xtick',round(nu(1:n/8:n),2))
set(gca,'ytick',(0:n/4:n))
```

```
yyhat=fftshift(ifft(fftshift(ff)));
figure;
plot(t,real(yyhat),'k','LineWidth',1.5)
xlim([0,dt*n]), ylim([5,15])
set(gca,'xtick',(0:dt*n/8:dt*n))
set(gca,'ytick',(5:2:15))
figure;
plot(t,imag(yyhat),'k','LineWidth',1.5)
xlim([0,dt*n])
set(gca,'xtick',(0:dt*n/8:dt*n))
```

Matlab’s function uses a fast FFT algorithm.
The FFT is a specific technique to implement the DFT.
You can use Matlab’s function.

2D Discrete Fourier Transform

We can also take the discrete Fourier transform of a 2D image.
We utilize the same discrete Fourier Matrices.

$$\underset{n_y \times n_x}{F} = \underset{n_y \times n_y}{\Omega} \; \underset{n_y \times n_x}{Y} \; \underset{n_x \times n_x}{\Omega'}$$

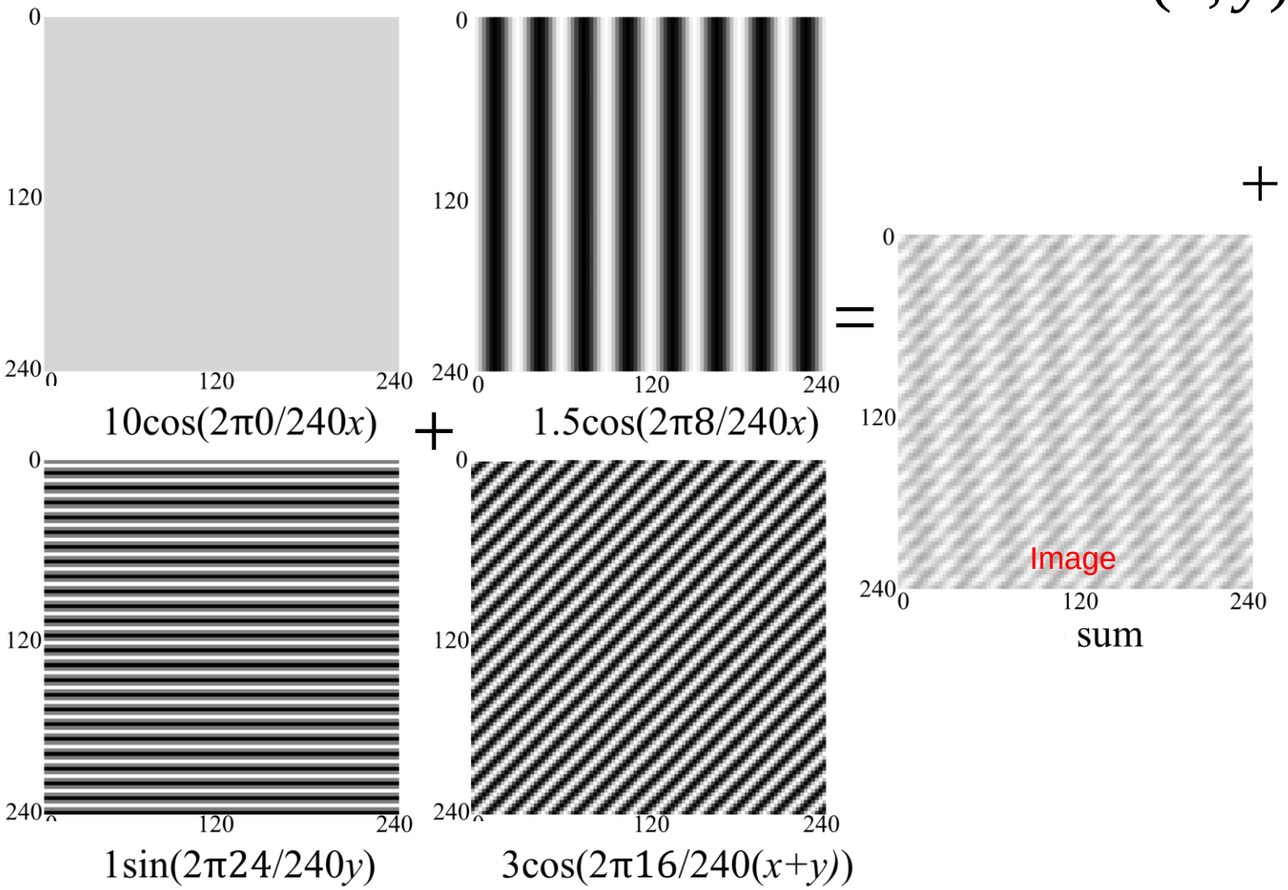
The result is an array of amplitude coefficients for the various frequencies.

Fourier

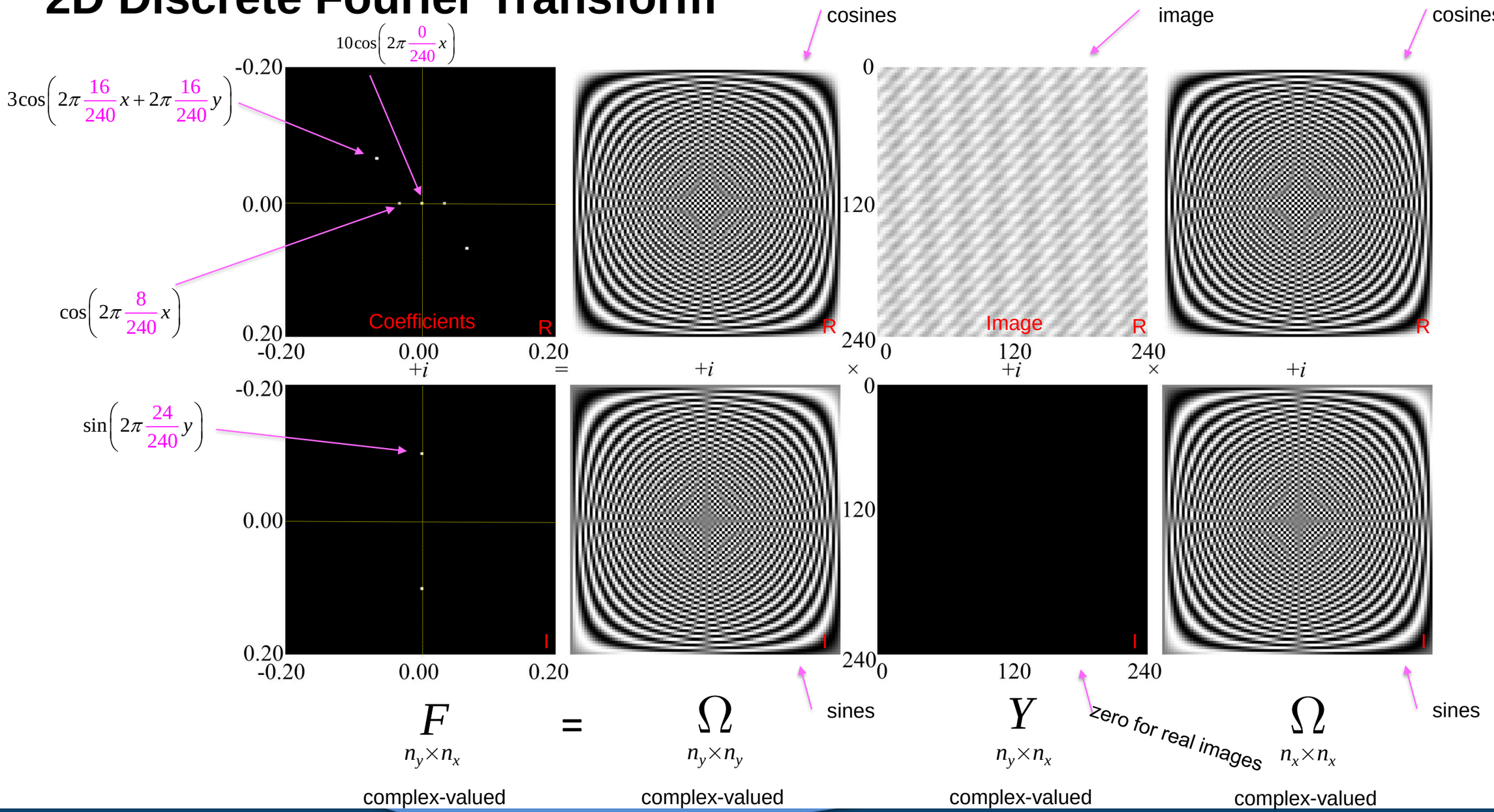
2D Discrete Fourier Transform

Example: Let's sample the continuous image scene (2D function)

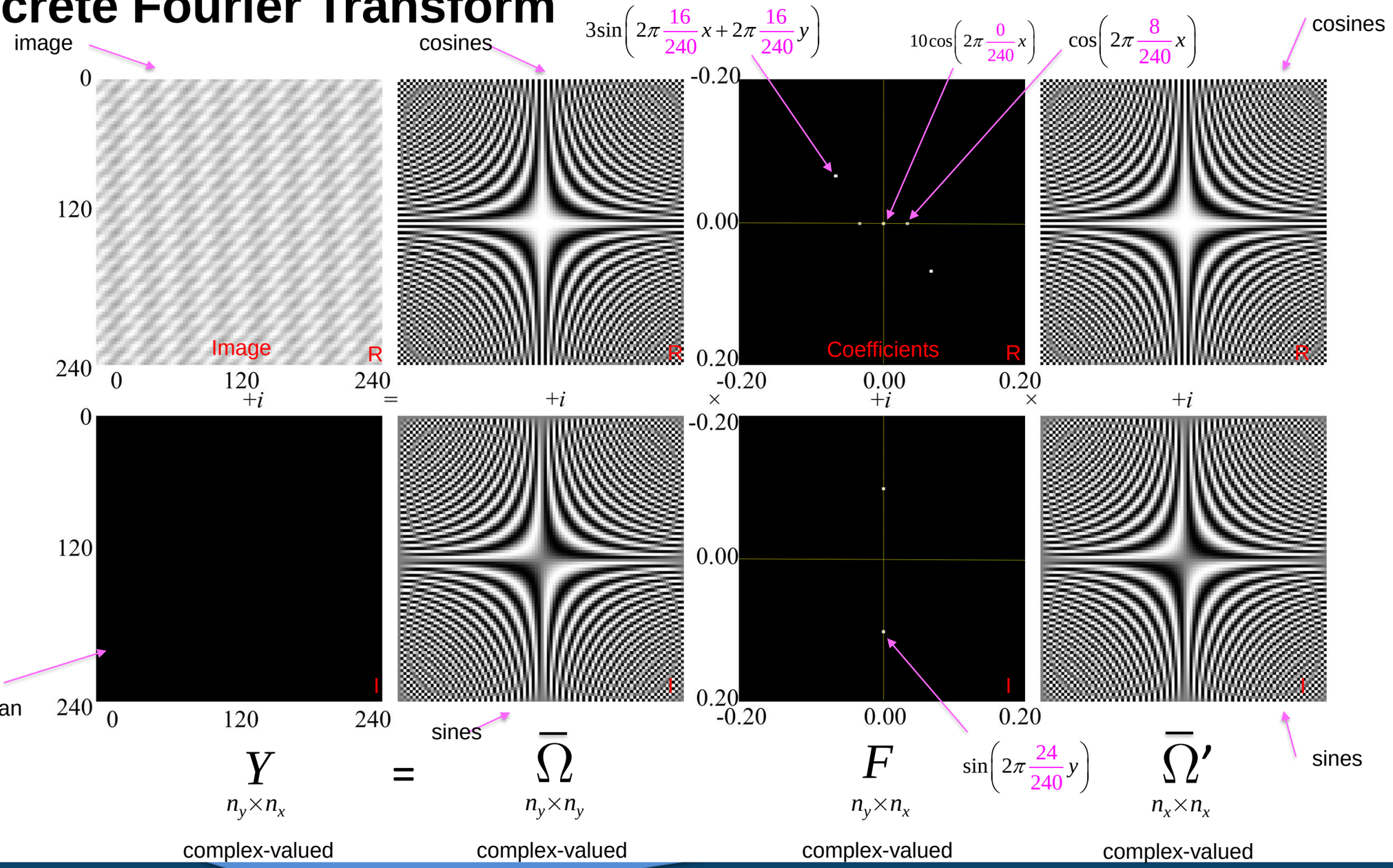
$$Y(x,y) = 10\cos\left(2\pi\frac{0}{240}x\right) + \frac{3}{2}\cos\left(2\pi\frac{8}{240}x\right) + \sin\left(2\pi\frac{24}{240}y\right) + \cos\left(2\pi\left(\frac{16}{240}x + \frac{16}{240}y\right)\right)$$



2D Discrete Fourier Transform



2D Discrete Fourier Transform

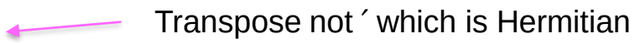


2D Discrete Fourier Transform

```
ny=96; nx=ny;

FOVy=240; FOVx=240; deltay=FOVy/ny; deltax=FOVx/nx;
deltakx=1/ny/deltax; deltaky=1/nx/deltay;
A1=10; A2=1.5; A3=1; A4=3;
nu1=0/240; nu2=8/240; nu3=24/240; nu4=16/240;

% make image data
Y1=zeros(ny,nx); Y2=zeros(ny,nx); Y3=zeros(ny,nx); Y4=zeros(ny,nx);
for k=1:ny
    for j=1:nx
        Y1(k,j)= A1*cos(2*pi*nu1*(j-1)*deltax);
        Y2(k,j)= A2*cos(2*pi*nu2*(j-1)*deltax);
        Y3(k,j)= A3*sin(2*pi*nu3*(k-1)*deltay);
        Y4(k,j)= A4*cos(2*pi*nu4*(j-1)*deltax+2*pi*nu4*(k-1)*deltay);
    end
end
Y=Y1+Y2+Y3+Y4;

figure;
imagesc(Y,[-15,15])
axis image, axis off, colormap(gray)
```

2D Discrete Fourier Transform

```
% DFT by matrix multiplication
OmegaxC=zeros(nx,nx);
for k=1:nx
   for j=1:nx
      OmegaxC(k,j)=exp(-1i*2*pi*(k-(nx/2+1))*(j-(nx/2+1))/nx);
   end
end
OmegayC=zeros(ny,ny);
for k=1:ny
   for j=1:ny
      OmegayC(k,j)=exp(-1i*2*pi*(k-(ny/2+1))*(j-(ny/2+1))/ny);
   end
end

F  = (OmegayC*Y*transpose(OmegaxC));
figure;
imagesc(real(F),[0,2*nx*ny/2])%A1*2*nx*ny/2
axis image, axis off, colormap(gray)
figure;
imagesc(abs(imag(F)),[0,nx*ny/2])%A3*nx*ny/2
axis image, axis off, colormap(gray)
```



2D Discrete Fourier Transform

```
% take inverse Fourier transforms
OmegaBarxC=zeros(nx,nx);
for k=1:nx
    for j=1:nx
        OmegaBarxC(k,j)=exp(1i*2*pi*(k-(nx/2+1))*(j-(nx/2+1))/nx);
    end
end
OmegaBarxC=OmegaBarxC/nx;
OmegaBaryC=zeros(ny,ny);
for k=1:ny
    for j=1:ny
        OmegaBaryC(k,j)=exp(1i*2*pi*(k-(ny/2+1))*(j-(ny/2+1))/ny);
    end
end
OmegaBaryC=OmegaBaryC/ny;

Yhat  = (OmegaBaryC*F*transpose(OmegaBarxC));

figure;
imagesc(real(Yhat),[-15,15])
axis image, colormap(gray), axis off
figure;
imagesc(imag(Yhat),[-15,15])
axis image, colormap(gray), axis off
```

2D Discrete Fourier Transform

Matlab has built in functions that perform the discrete 2D Fourier transform.

```
FF=fftshift(fft2(fftshift(Y)));
figure;
imagesc(real(FF),[0,2*nx*ny/2])%A1*2*nx*ny/2
axis image, axis off, colormap(gray)
figure;
imagesc(abs(imag(FF)),[0,nx*ny/2])%A3*nx*ny/2
axis image, axis off, colormap(gray)
YYhat=fftshift(ifft2(fftshift(FF)));
figure;
imagesc(real(Yhat),[-15,15])
axis image, axis off, colormap(gray)
figure;
imagesc(imag(Yhat),[-15,15])
axis image, axis off, colormap(gray)
```

Matlab's function uses a fast FFT algorithm.

The FFT is a specific technique to implement the DFT.

You can use Matlab's function.

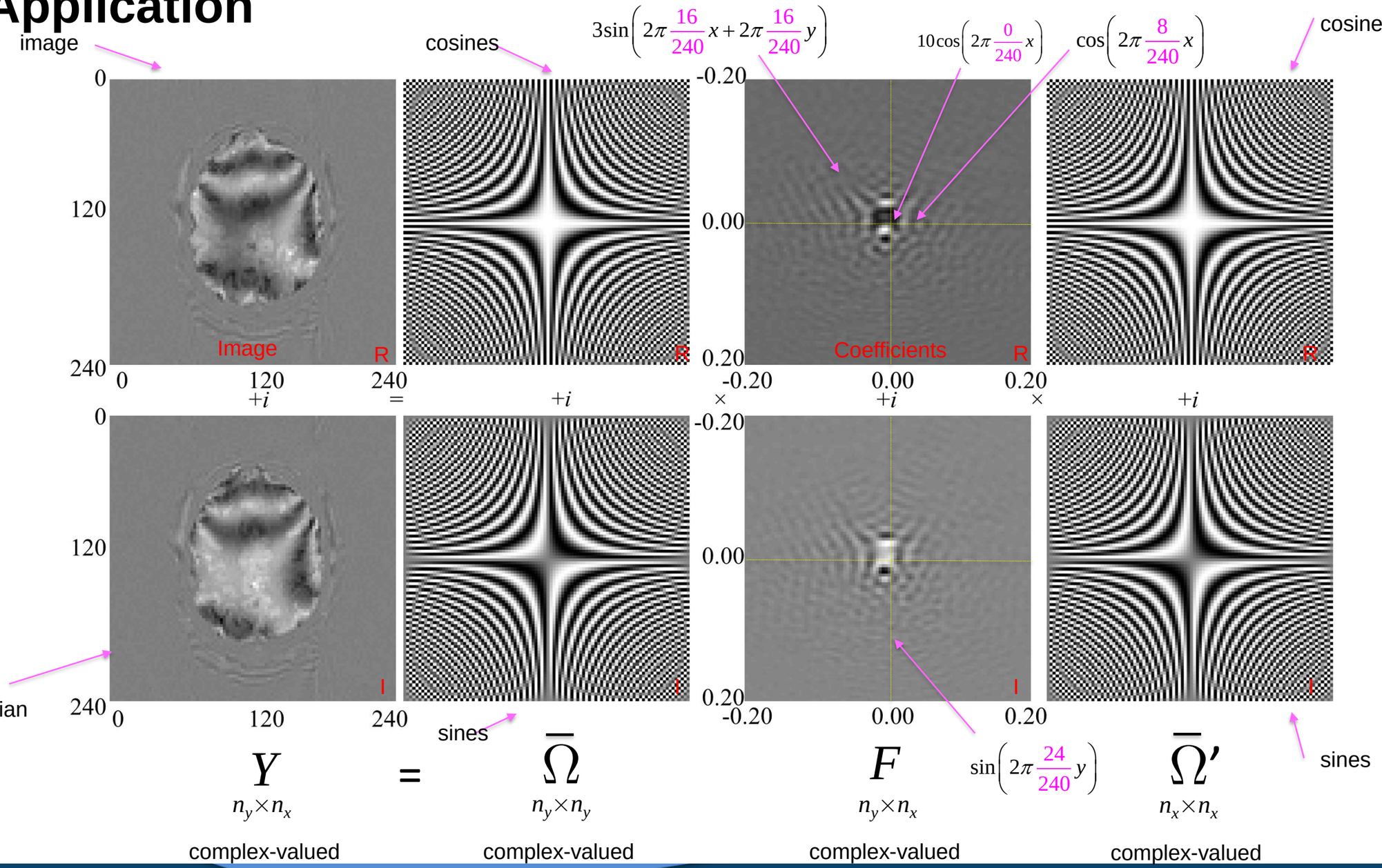
FMRI Application

In MRI and FMRI, the “spatial frequencies” are what is measured and an image is produced by taking the 2D inverse discrete Fourier transform.

This is generally an involved process due to systematic measurement inconsistencies.

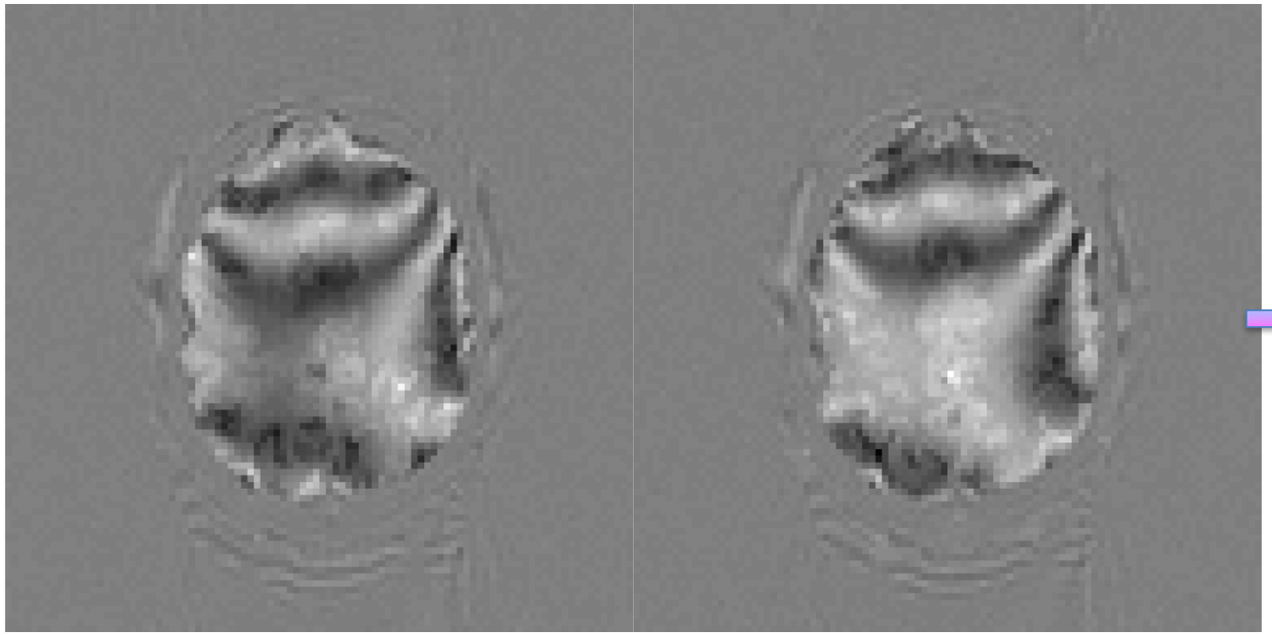
But once inconsistencies have been compensated, it is straightforward.

FMRI Application



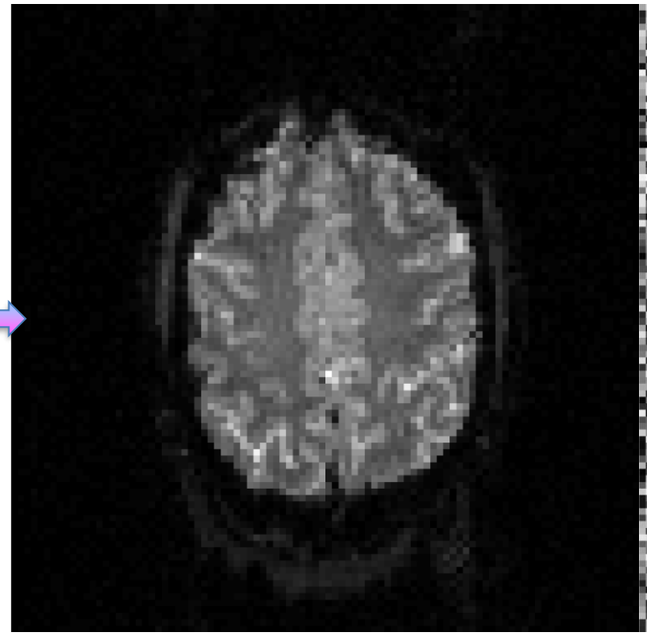
FMRI Application

Conversion from Cartesian real-imaginary coordinated to polar magnitude and phase.

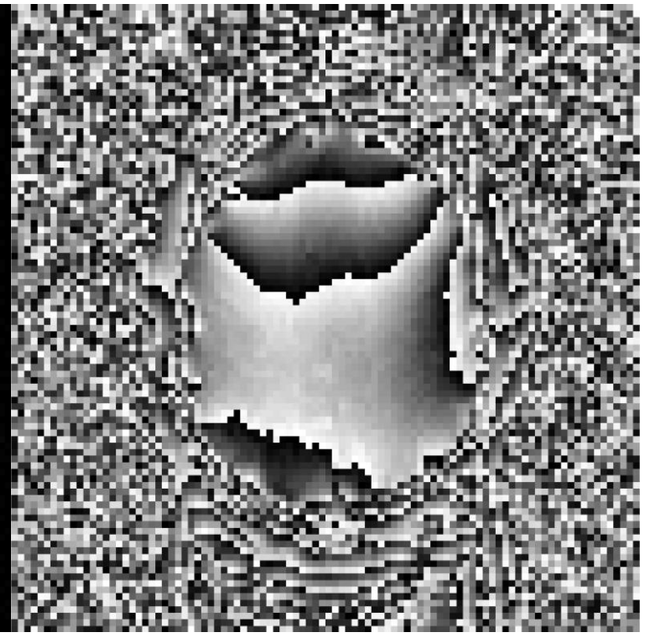


Real

Imaginary



Magnitude



Phase

FMRI Application

Matlab code to inverse IDT reconstruct image FMRI image.

```
load F.mat

figure;
subplot(1,2,1)
imagesc(real(F))
axis image, axis off, colormap(gray)
subplot(1,2,2)
imagesc(imag(F))
axis image, axis off, colormap(gray)
figure;
subplot(1,2,1)
imagesc(abs(F))
axis image, axis off, colormap(gray)
subplot(1,2,2)
imagesc(angle(F),[-pi,pi])
axis image, axis off, colormap(gray)
```

```
Y=fftshift(ifft2(fftshift(F)));

figure;
subplot(1,2,1)
imagesc(real(Y),[-7.5,7.5])
axis image, colormap(gray), axis off
subplot(1,2,2)
imagesc(imag(Y),[-7.5,7.5])
axis image, colormap(gray), axis off
figure;
subplot(1,2,1)
imagesc(abs(Y),[0,7.5])
axis image, colormap(gray), axis off
subplot(1,2,2)
imagesc(angle(Y),[-pi,pi])
axis image, colormap(gray), axis off
```

Discussion

Some basic mathematics of the discrete Fourier transform was described.

The DFT was calculated using a complex-valued matrix representation.

The DFT was recalculated using built-in Matlab functions.

A quick FMRI application was described.

Coming up: Convolution via the DFT.

Discussion

Questions?

Homework 7

1. Load your own time series into Matlab and calculate 1D DFT.
   Plot the original time series and real and imaginary parts of 1D DFT.

2. Load your own image into Matlab and calculate 2D DFT.
   Make images of original image and real and imaginary parts of 2D DFT.

3. Make a new spatial frequency array that is only the center $n_y/2 \times n_x/2$
   portion of your array in #2. IFT with reduced dimension size.

4*.Set some spatial frequencies from your image to zero and IFT.
   Make an image.

*For students in MSSC 5770.

Homework 7

Submit one document with all your results (no Matlab code) and some discussion of thoughts or commentary.
Separately also submit executable Matlab code and any needed files.