# Transform Methods & Signal Processing

## lecture 04

Matthew Roughan
<matthew.roughan@adelaide.edu.au>

Discipline of Applied Mathematics
School of Mathematical Sciences
University of Adelaide

October 13, 2009

This lecture extends our work on the DFT into 2D. We see that the DFT naturally generalizes to higher dimensions, and this can be very useful in image processing.

# Transforms in 2D

Until now we have only considered 1D functions as possible inputs, but there are many applications where we want to consider functions with 2 (or more) independent variables, e.g., surfaces, and the Fourier transform naturally generalizes to this case.

# Transforms in 2D

We want a separable basis.

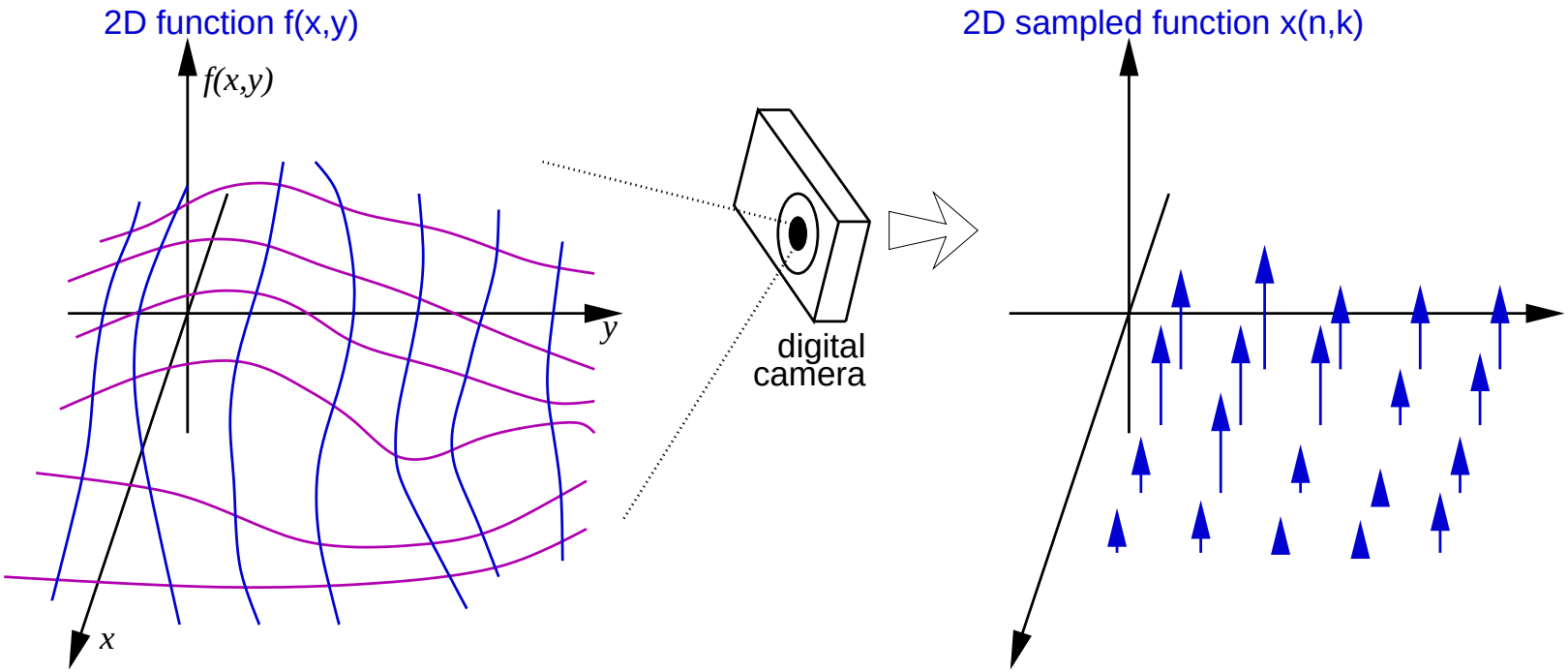
- basis "vectors" are the product of basis vectors from two component subspaces. e.g. if the function can be written as $f(x,y) = g(x)h(y)$, we should be able to write its FT as $F(s,t) = G(s)H(t)$.
- makes computation easier
- makes most of the math easier (same methods used for proofs)

Fourier transform in 2D

$$F(s,t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-i2\pi(sx+ty)} \, dx \, dy$$

# 2D signals = images

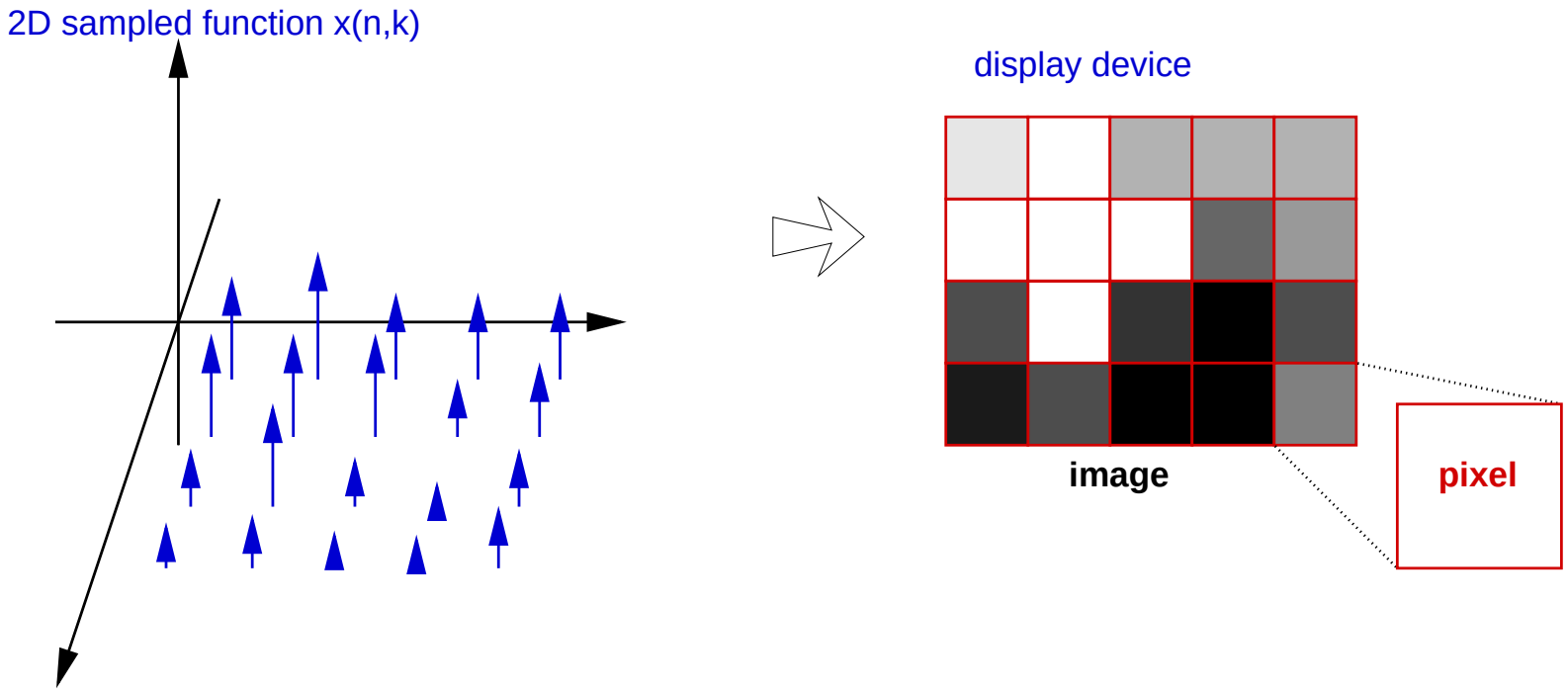
- 2D signal processing is used for images
- images are sampled signals in 2D




- same issues for sampling/quantization as we have for 1D signals

# Displaying 2D signals

- 2D signal processing is used for images
- images are sampled signals in 2D



I will show large values using pale, or white pixels, and dark pixels will indicate small (or very negative) values.

# The DFT in 2D

DFT

$$X(k_1,k_2) = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x(n_1,n_2)e^{-i2\pi k_1 n_1/N_1}e^{-i2\pi k_2 n_2/N_2},$$

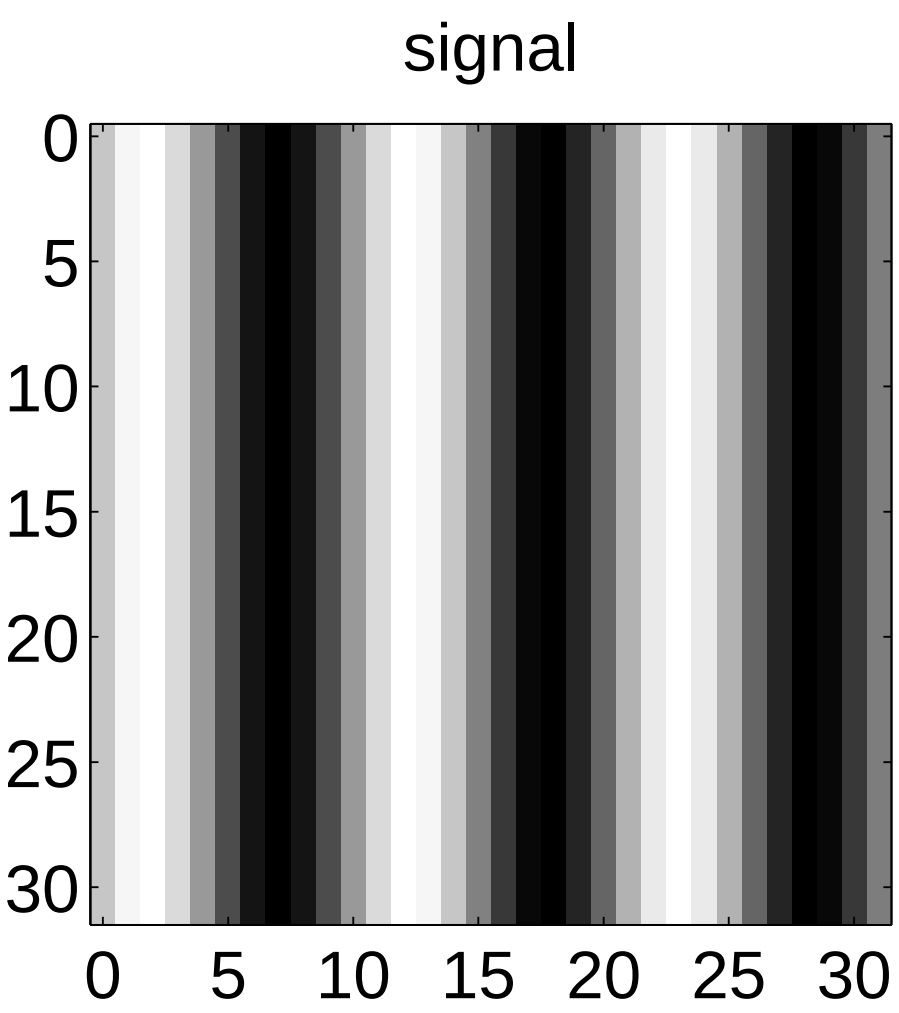
- To compute it efficiently:
  1. compute 1D FFT along the rows
  2. then do a 1D FFT along the columns
- Called **row-column** algorithm
  - note that the order could change.
- naturally generalizes to higher dimensions

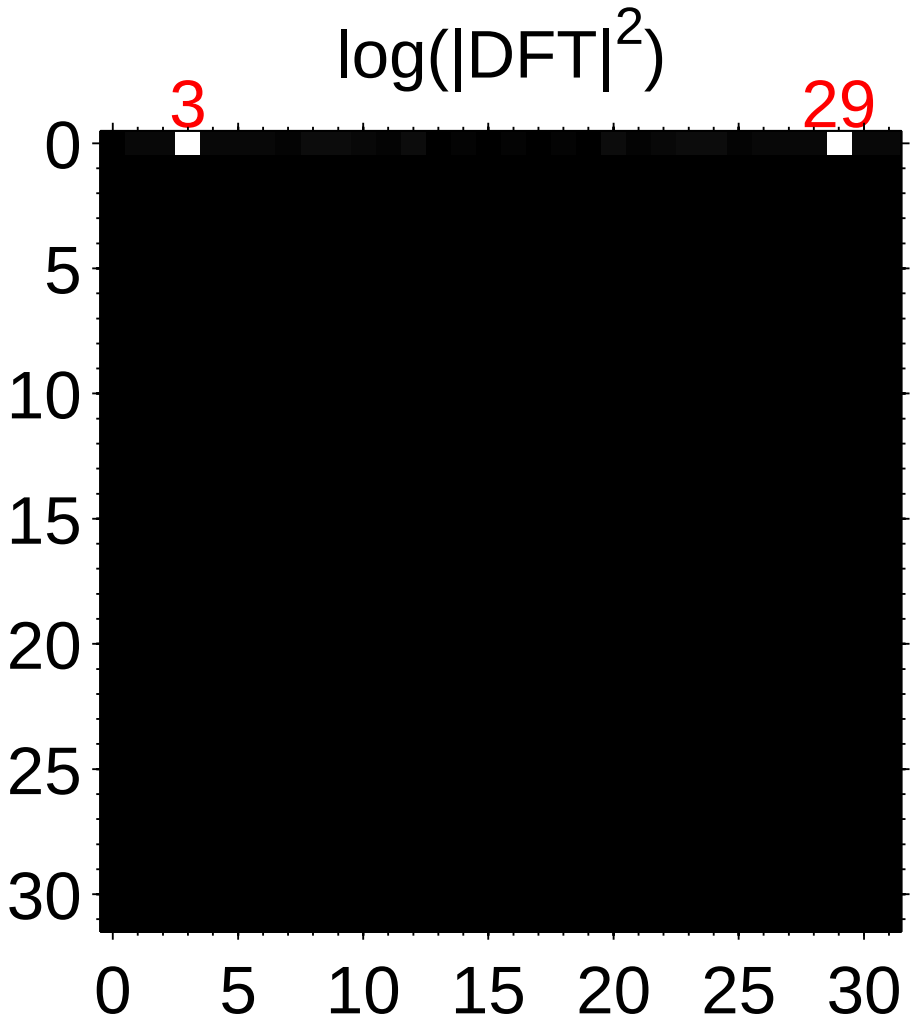
Computations in the row-column algorithm, assume image is $N_1 \times N_2 = N$ pixels in size. Computations will be

$$\frac{N}{N_1} O(N_1 \log N_1) + \frac{N}{N_2} O(N_2 \log N_2) = O(N \log N_1 + N \log N_2) = O(N \log[N_1 \times N_2]) = O(N \log N)$$

# Examples (i)

$$x(n,k) = \sin(2\pi \mathbf{3}k/N)$$



In the images displayed, light colors indicate large values, and dark colors indicate small values.

Note that the image has a frequency of 3 along the $x$-axis (as $k$ varies), and it is constant along the $y$-axis (as $n$ varies). Note the change in sense, Matlab uses matrix notation, where the first index is the row, and the second the column.

In the frequency domain the power-spectrum has two peaks, both at 0 frequency in the $y$ direction, and one at frequency 3 in the $x$ direction. The second peak is the symmetric peak (see previous 1D examples) at frequency $32 - 3 = 29$. We could use fftshift here as well to make these symmetric about the center of the graph.

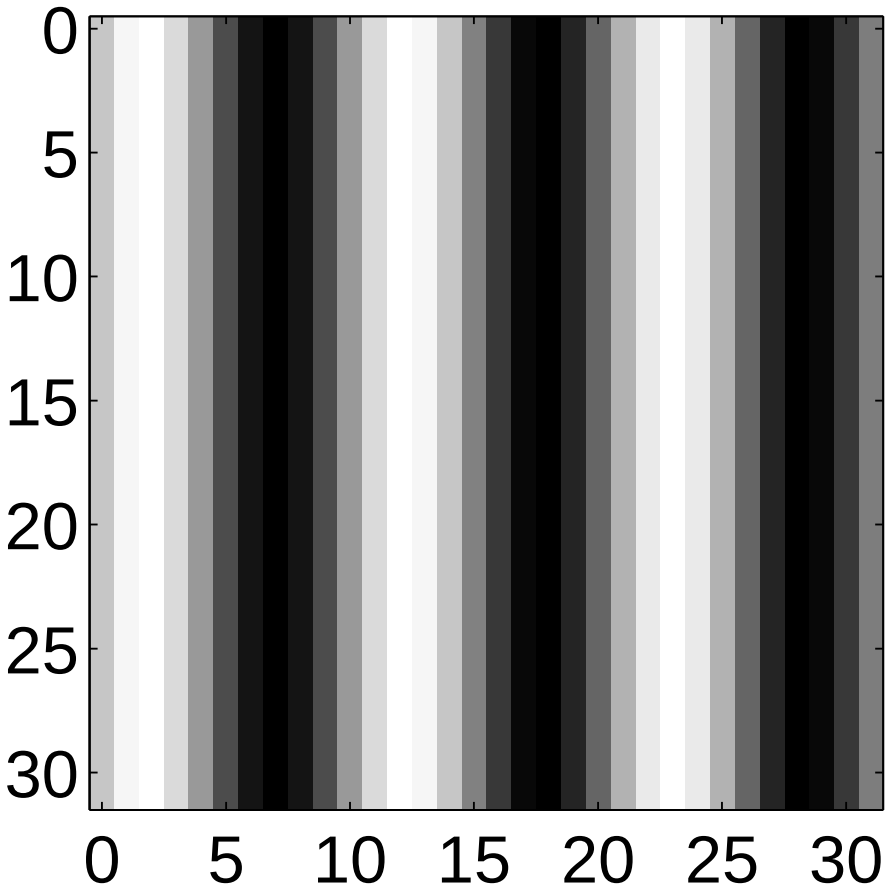
Take the signal (in the left hand graph) to be $x$, and the right hand graph plots the power-spectrum $\log(|X|^2)$.

Finally note that in the image, we have indices that run from $0, \ldots, 31$, whereas in Matlab, indices would run from $1, \ldots, 32$. Either is acceptable nomenclature (we use $0, \ldots, 31$ here because it is easier to interpret frequencies with these indices).

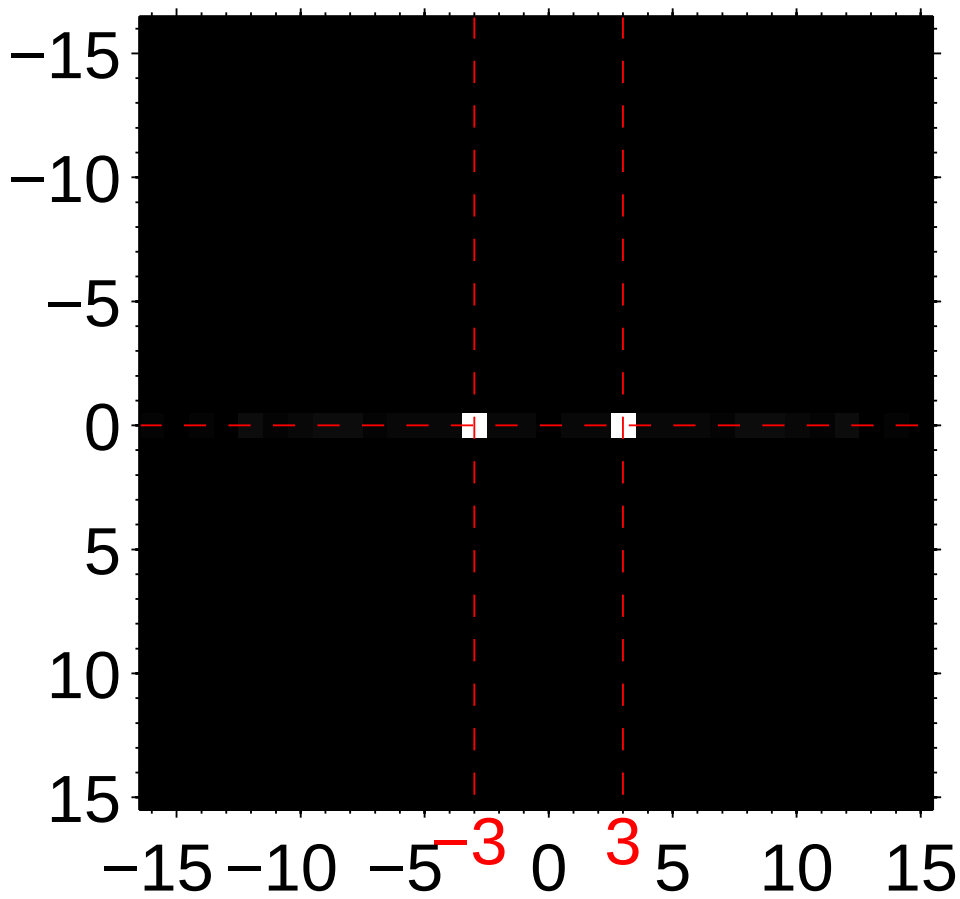
# Examples (i): `fftshift`

$$x(n,k) = \sin(2\pi \mathbf{3}k/N)$$

signal




log(|DFT|$^2$)

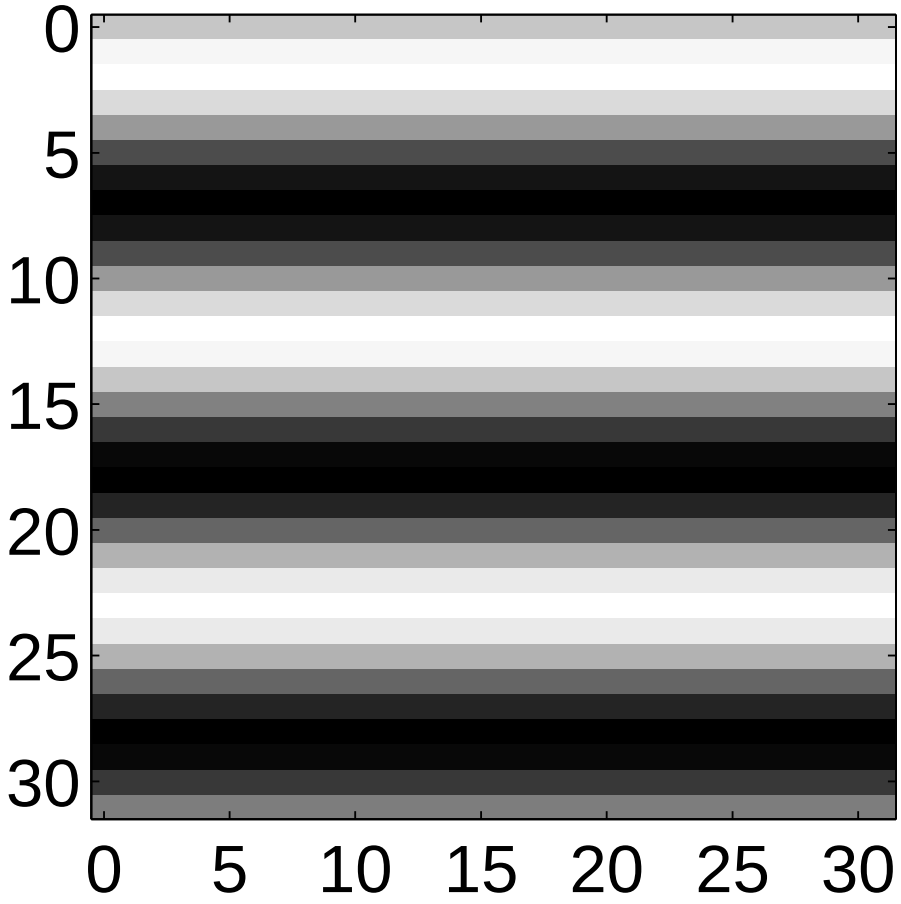


When fftshift is applied the $(0,0)$ frequency is shifted to the center of the image. Now the peaks are more obviously the result of a simple sinusoidal function (compare to the 1D power-spectrum of sin).

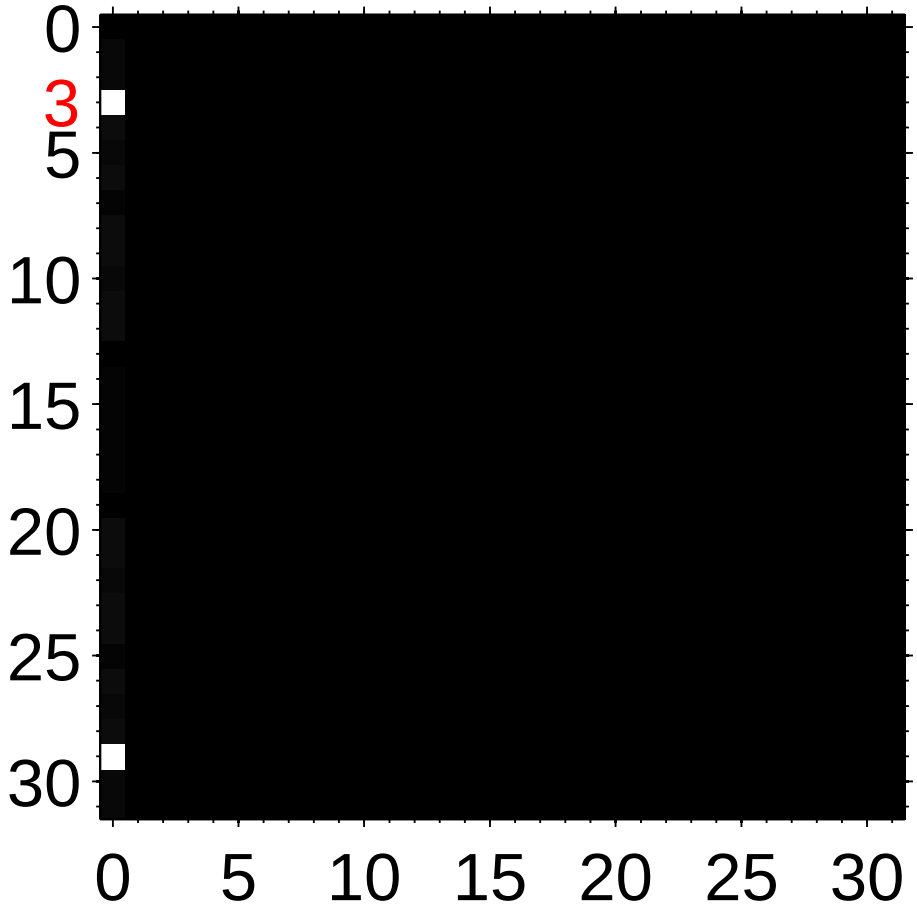
# Examples (ii)

$$x(n,k) = \sin(2\pi 3n/N)$$

signal



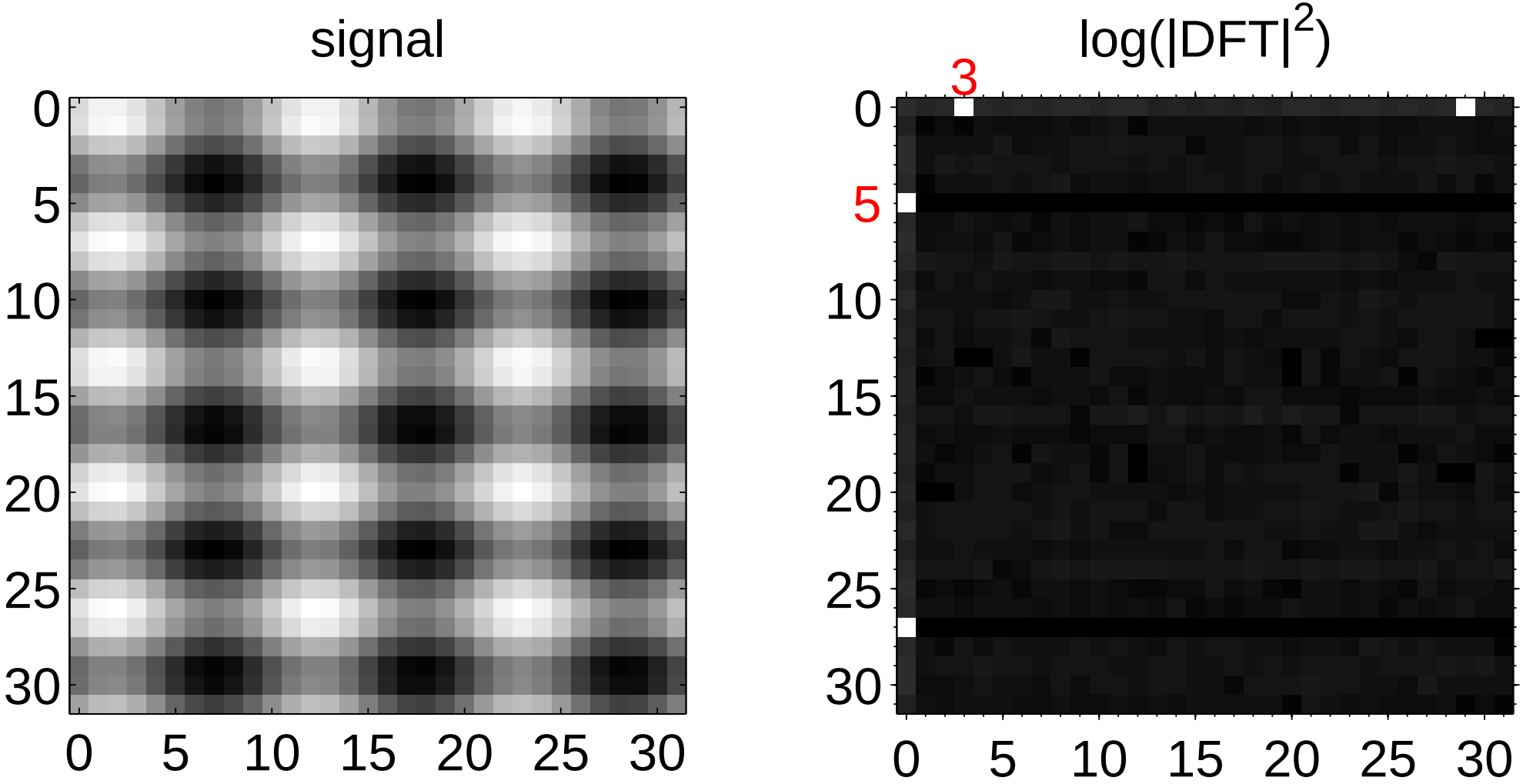

log(|DFT|$^2$)



Here the periodicity is in the $y$-axis, and we see a flip of the previous graph, so that the frequency peaks occur at zero frequency for $x$-direction.

# Examples (iii): superposition

$$x(n,k) = \sin(2\pi \mathbf{5}n/N) + \sin(2\pi \mathbf{3}k/N)$$

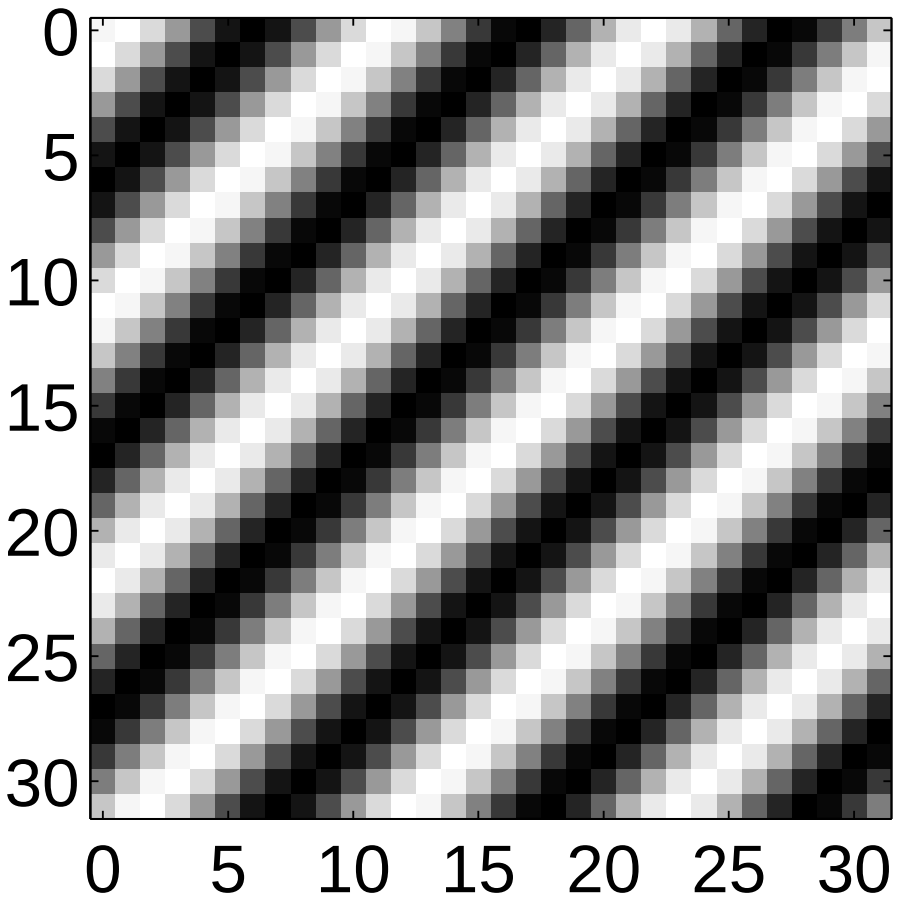


Remember the Fourier transform is linear, so when we superpose two simple sinusoids (by adding them), we simply add the corresponding Fourier transforms, so this picture shows peaks corresponding to the two sinusoids one in each direction.

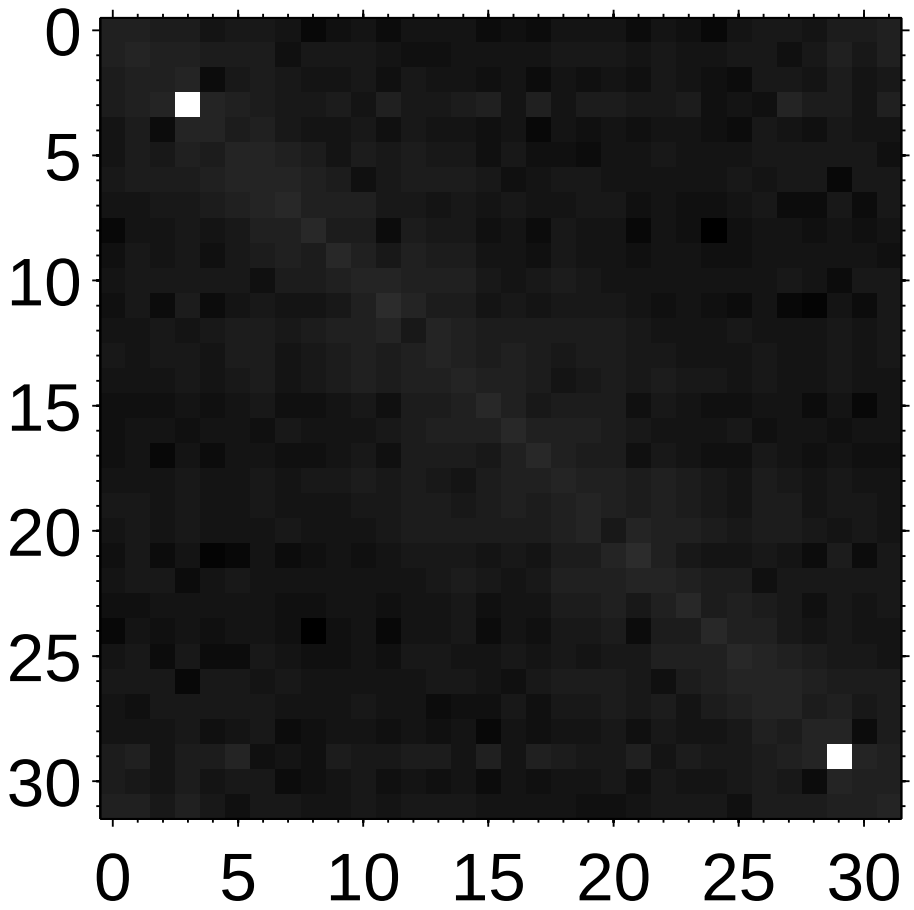
# Examples (iv)

$$x(n,k) = \sin(2\pi \mathbf{3}(n+k)/N)$$

signal




log(|DFT|$^2$)



Here we have a single sine function, and so we see only two peaks in the frequency domain, but they are in more complex positions. We will see how to understand this by understanding how symmetries work in 2D.

# DFT and symmetry

The symmetry of the 2D FT depends on the symmetry of the function.

$$\begin{aligned} F(-s,-v) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{i2\pi(sx+ty)}\,dx\,dy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(-x,-y)e^{-i2\pi(sx+ty)}\,dx\,dy \\ &= \mathcal{F}\{f(-x,-y)\} \end{aligned}$$

As before (in 1D), but now we reflect through the origin.

- similar result to before relating complex conjugates etc.

# DFT and symmetry

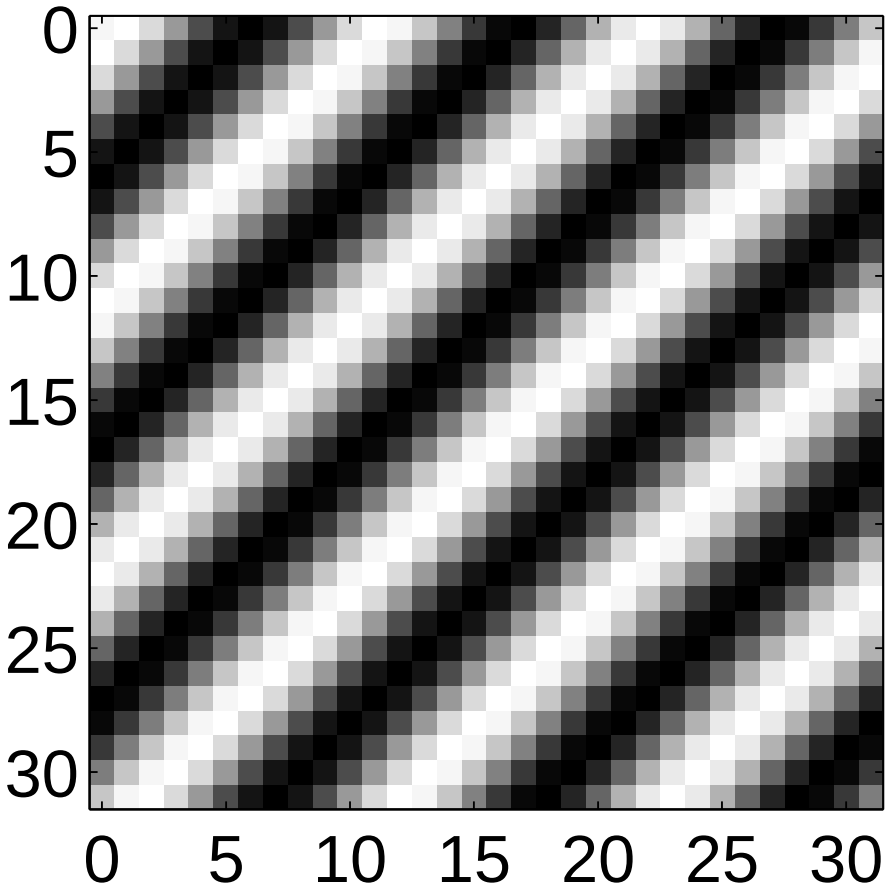
Power-spectrum of 2D DFT will be symmetric about the center (zero frequency).

- Equivalent to real time series produces even power-spectrum.
- In matlab, use `fftshift` to see the plots this way.

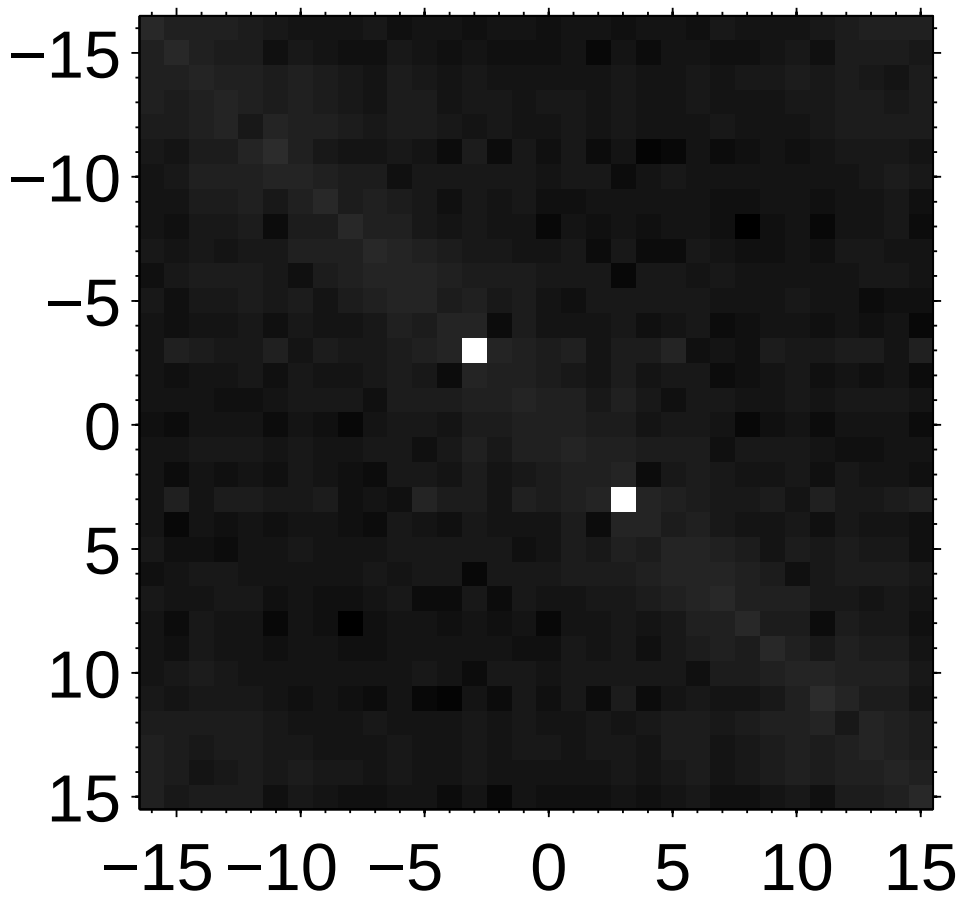
# Examples (iv-b)

as before using `fftshift`

signal




$\log(|\mathrm{DFT}|^2)$



# 2D DFT of a sinusoid

$$f(x,y) = \cos\left[2\pi s_0(\cos(\theta)x + \sin(\theta)y)\right]$$

Multiply by $g_\tau(u)g_\tau(v) =$ Gaussians with standard dev. $\tau$, where $u$ an $v$ are given in the coord. transform below.

$$\begin{aligned} F(s,t) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g_\tau(u)g_\tau(v)\, f(x,y)e^{-i2\pi(sx+ty)}\,dx\,dy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g_\tau(u)g_\tau(v)\cos\left[2\pi s_0(\cos(\theta)x + \sin(\theta)y)\right] e^{-i2\pi(sx+ty)}\,dx\,dy \end{aligned}$$

Change co-ordinates, so that $u = \cos(\theta)x + \sin(\theta)y$ and $v = -\sin(\theta)x + \cos(\theta)y$, and the transformation is just a rotation so the Jacobian is $J = \begin{vmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{vmatrix} = 1.$

# 2D DFT of a sinusoid

Note $g(r)$ remains unchanged by a rotation in the co-ordinates.

$$F(s,t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g_\tau(u)g_\tau(v)\cos\left[2\pi s_0 u\right] e^{-i2\pi u(\cos(\theta)s+\sin(\theta)t)} e^{-i2\pi v(-\sin(\theta)s+\cos(\theta)t)}\, du\, dv$$

$$= \int_{-\infty}^{\infty} g_\tau(v) e^{-i2\pi v(-\sin(\theta)s+\cos(\theta)t)}\, dv \int_{-\infty}^{\infty} g_\tau(u)\cos\left[2\pi s_0 u\right] e^{-i2\pi u(\cos(\theta)s+\sin(\theta)t)}\, du$$

The second integral is just the Fourier transform of a cosine (with a Gaussian), so as the Gaussian width increases $\tau \to \infty$, it becomes the FT of a cosine.

$$\frac{1}{2}\left[\delta(\cos(\theta)s+\sin(\theta)t-s_0)+\delta(\cos(\theta)s+\sin(\theta)t+s_0)\right]$$

A pair of parallel "ridges" in the $(s,t)$ plane.

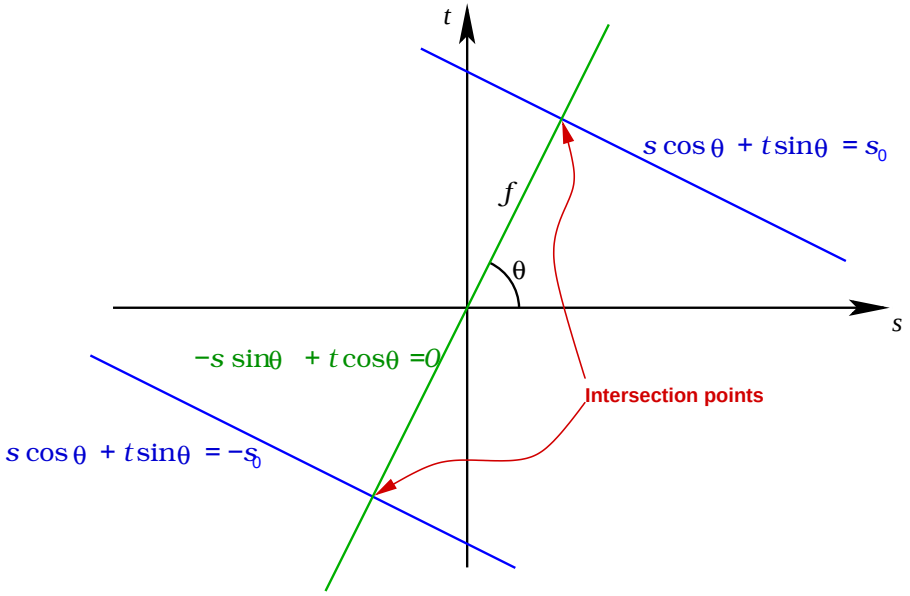
# 2D DFT of a sinusoid

## Result

$$F(s,t) = \int_{-\infty}^{\infty} e^{-i2\pi v(-\sin(\theta)s+\cos(\theta)t)}\,dv \int_{-\infty}^{\infty} \cos\left[2\pi s_0 u\right] e^{-i2\pi u(\cos(\theta)s+\sin(\theta)t)}\,du$$

$$= \frac{1}{2}\delta(-\sin(\theta)s+\cos(\theta)t)\left[\delta(\cos(\theta)s+\sin(\theta)t-s_0)+\delta(\cos(\theta)s+\sin(\theta)t+s_0)\right]$$

A pair of parallel ridges, at right angles to another ridge, intersecting at two points.



# 2D DFT of a sinusoid

$f(x,y) = \cos[2\pi s_0(\cos(\theta)x + \sin(\theta)y)]$ represents a sinusoid rotated by $\theta$ around the center of the plane.

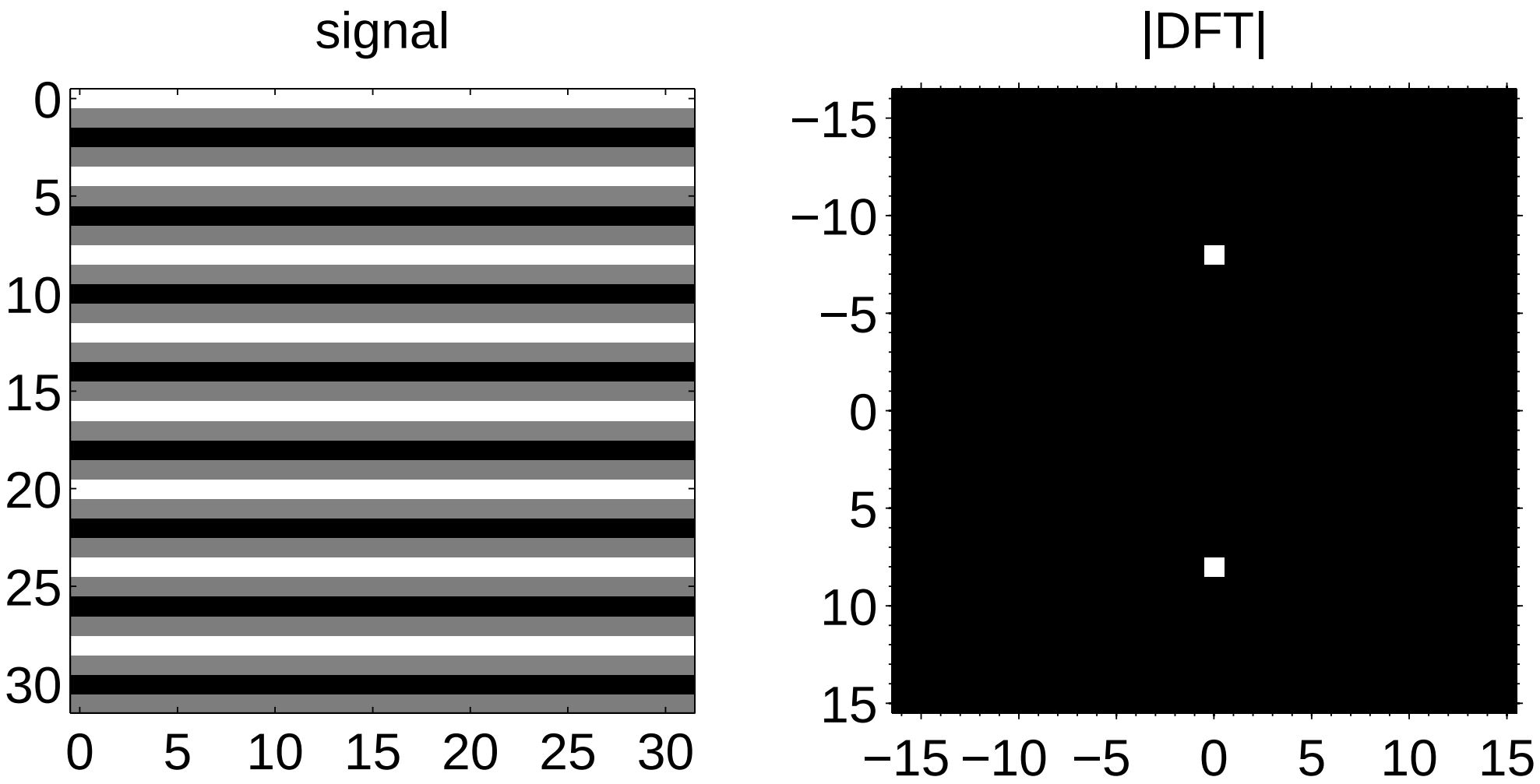
We have seen that the result is that the FT is also rotated by $\theta$, about the center.

This holds more generally – a spatial rotation causes a rotation in the frequency domain.

However, its not quite that simple for discrete transforms, where there is no such thing as an exact rotation (except by $k\pi/2$).

# Examples (v)

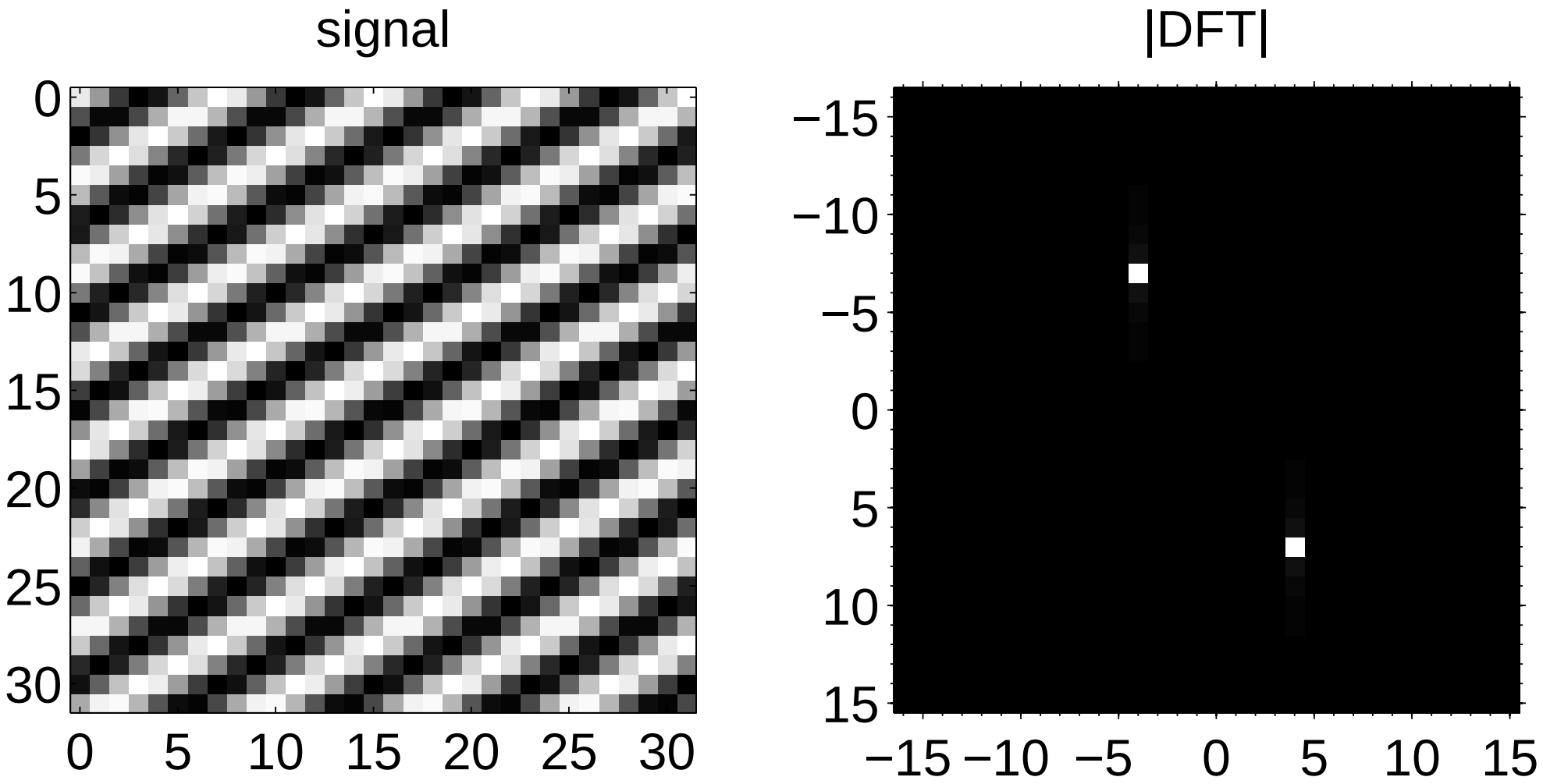
$x(n,k) = \sin(2\pi 8n/N)$ with `fftshift`



NB: the above plot is no longer on a log-scale so we can keep track of the peaks a bit more clearly.

# Examples (v-b)

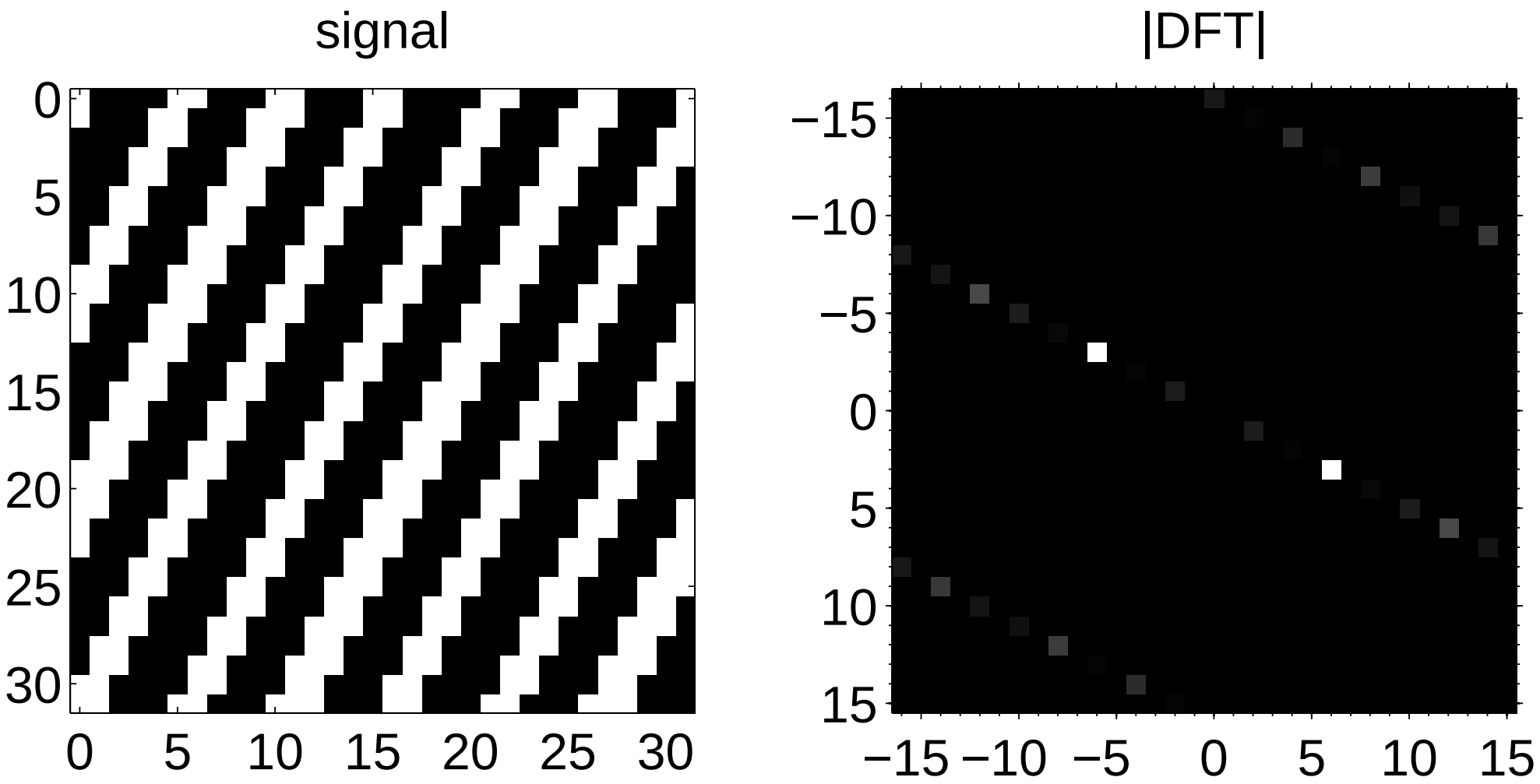
$x(n,k) = \sin(2\pi 8(\cos(\theta)n + \sin(\theta)k)/N)$ with `fftshift`



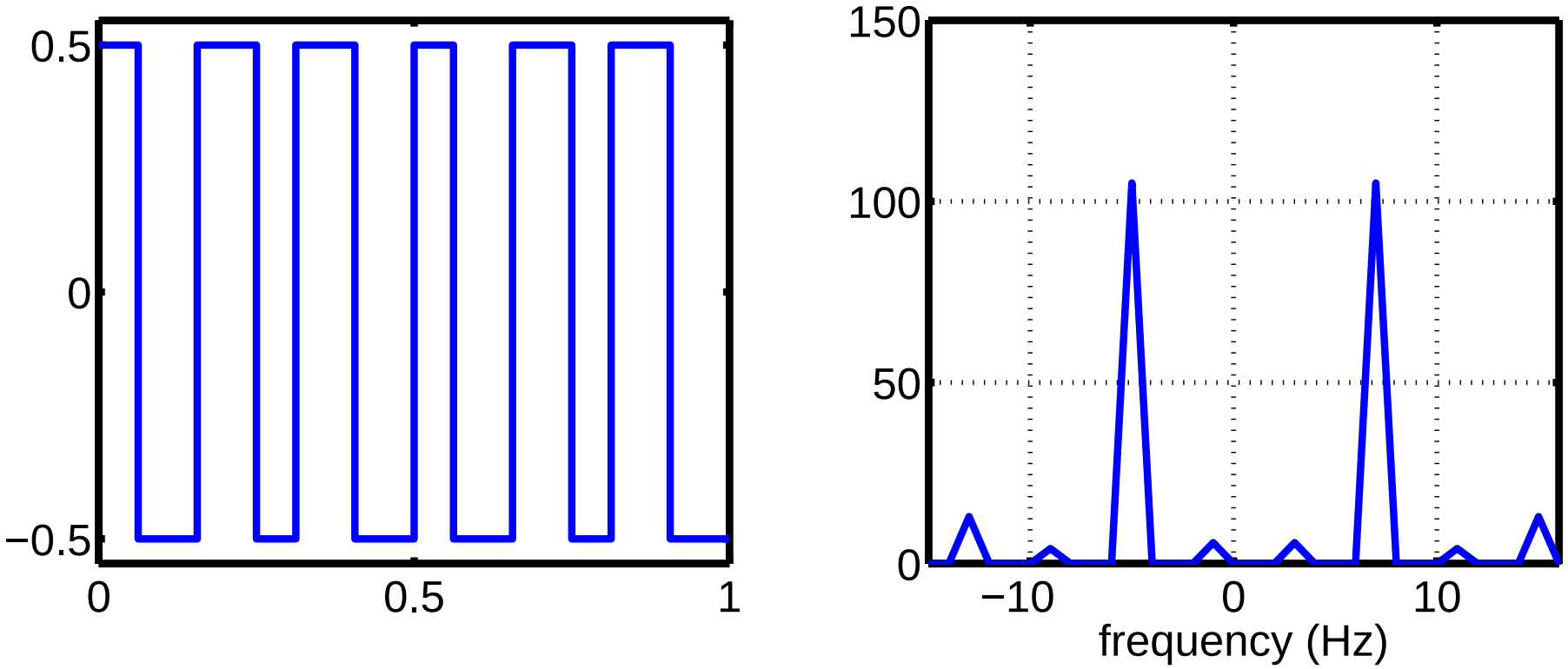
We can see that the FT looks like the FT of Example (v) rotated by the same amount as the signal.

# Examples (vi)

$x(n,k) = I\{\sin(2\pi(n+2k)/N) > 0.2\}$ with `fftshift`



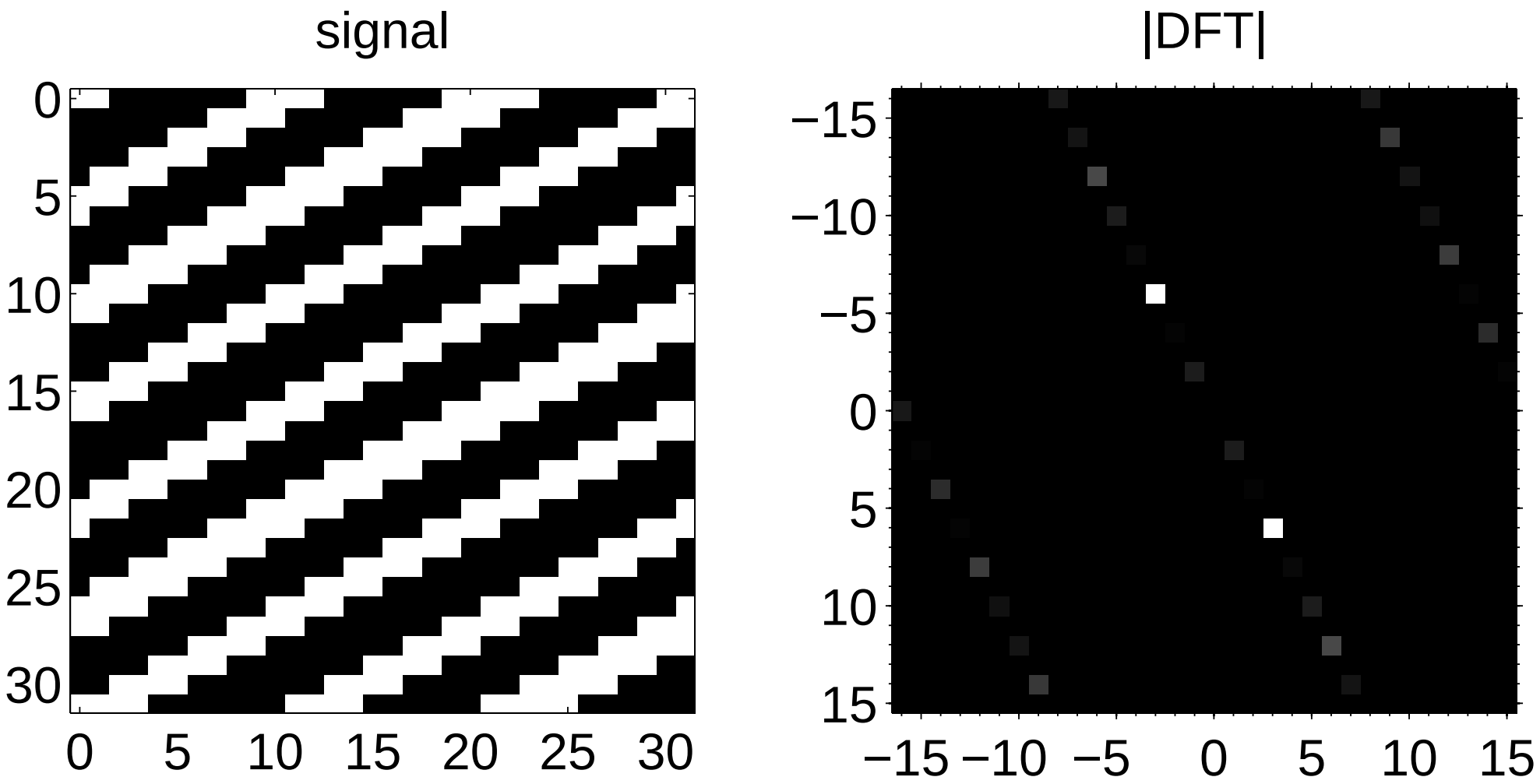
Until now, all of the signals have been a simple sine function. Now, we look at the equivalent of a square wave in 2D. We can easily work out the FT in 2D. Just consider the 1D FT of a square wave, and then rotate it (see figure below).




Note that $I(\cdot)$ is an indicator function (it is one, when the argument is true, and 0 otherwise).

# Examples (vii)

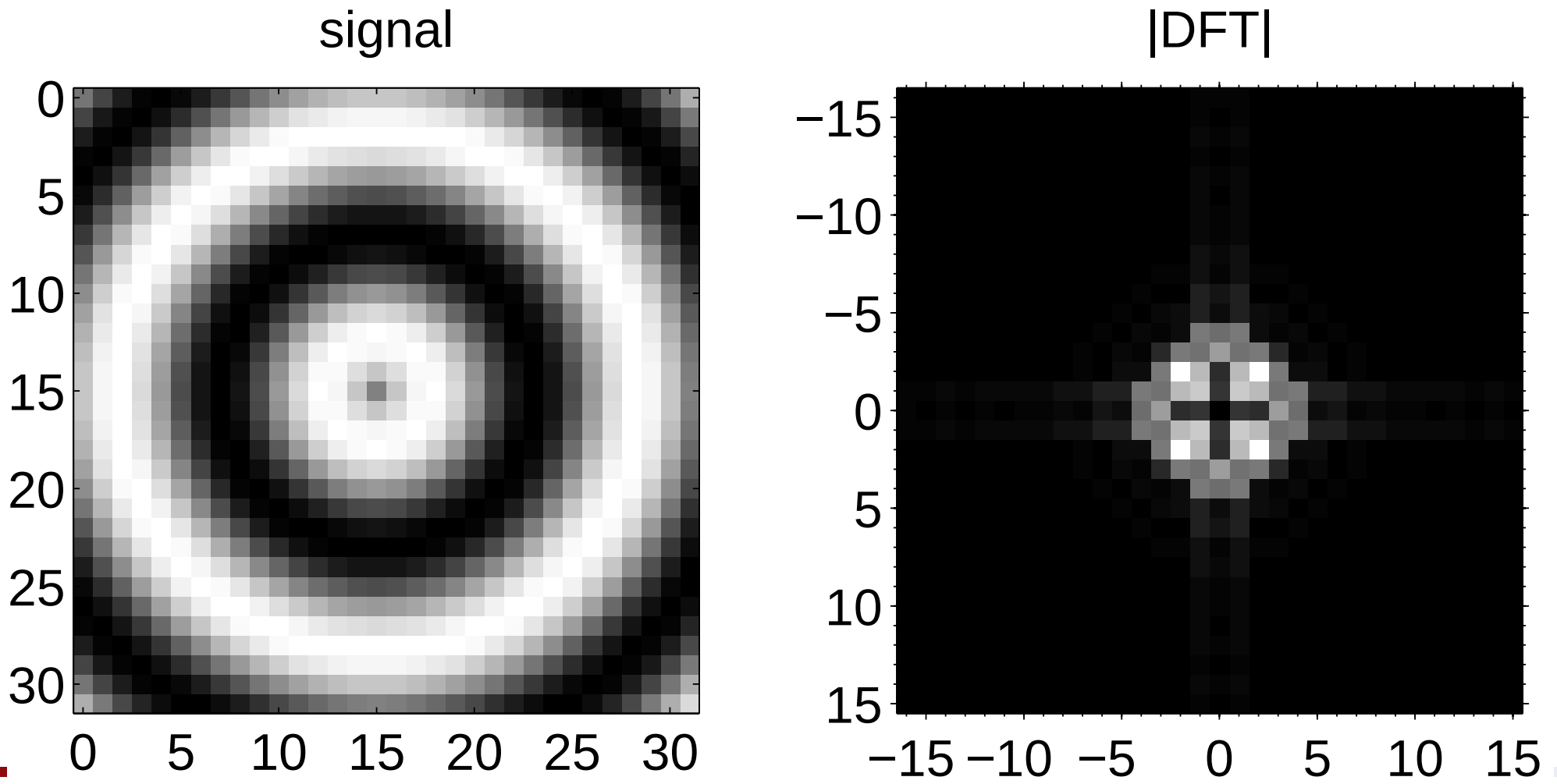
$x(n,k) = I\{\sin(2\pi(2n+k)/N) > 0.2\}$ with `fftshift`



```
%
% file:       two_d_examples_vii.m, (c) Matthew Roughan, Mon Aug  9 2004
%
colormap(gray)

N = 32;
x = (1:N)/N;
[X, Y] = meshgrid(x,x);

f_0 = 3;
A = sin(2*pi*f_0*(X+2*Y)) > 0.2;
A = A - mean(mean(A));
B = fftshift(fft2(A));

figure(7)
subplot(1,2,1)
image(A, 'CDataMapping', 'scaled');
set(gca, 'xtick', [0:5:32], 'fontsize', 18);
title('signal');

figure(7)
subplot(1,2,2)
image(abs(B), 'CDataMapping', 'scaled');
colormap(gray);
set(gca, 'xtick', [0:5:32], 'fontsize', 18);
title('abs(DFT)');

set(gcf, 'PaperUnits', 'centimeters', 'PaperPosition', [0 0 23 10])
print('-depsc', sprintf('Plots/2d_7.eps', i));
```

# Examples (viii)

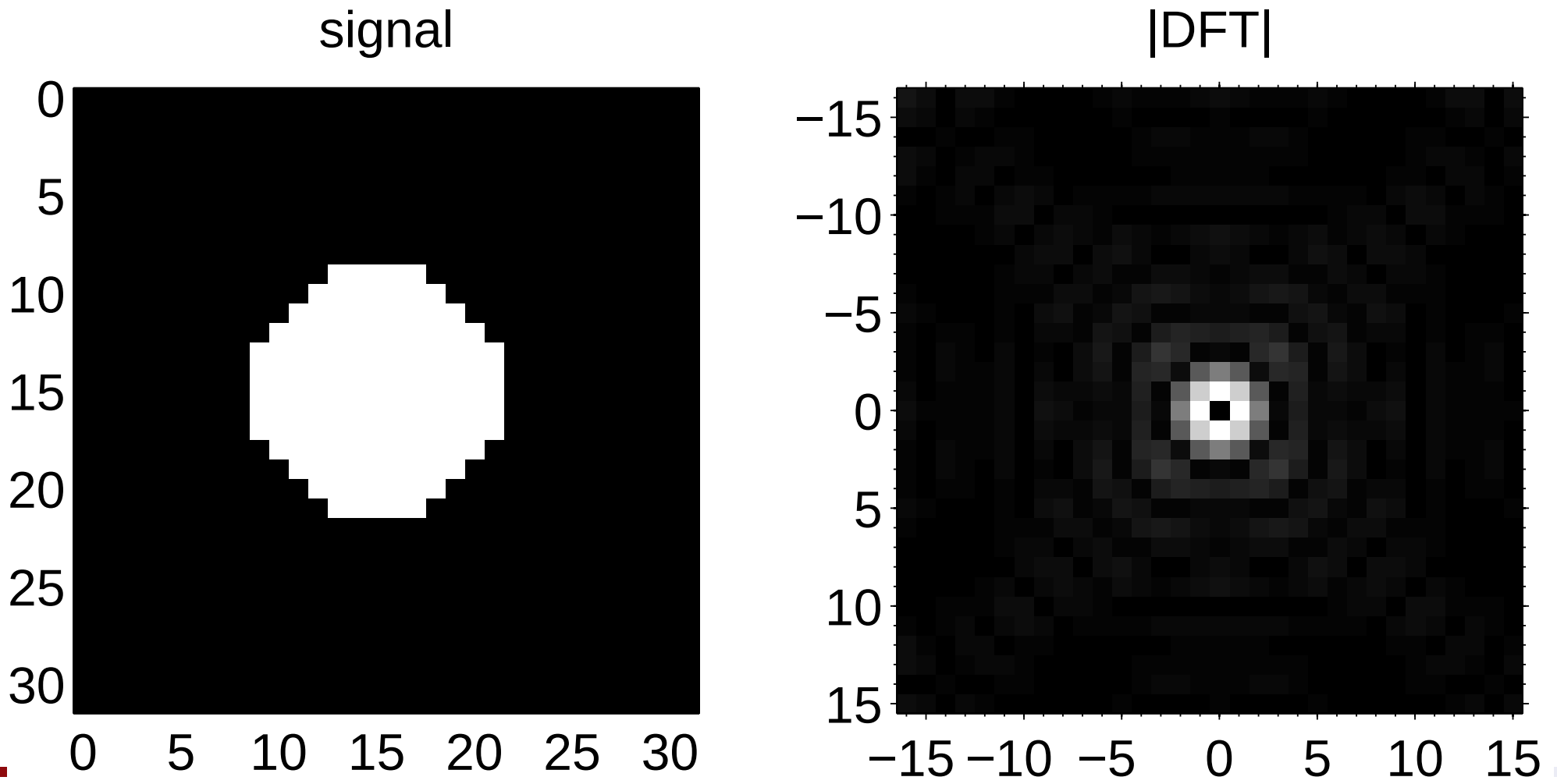
$$x(n,k) = \sin\left(2\pi\sqrt{(n/N-1/2)^2+(k/N-1/2)^2}\right)$$ with `fftshift`



This new signal has radial (or rotational) symmetry, i.e., it is invariant when we rotate it. Note that the FT is approximately rotationally invariant as well. The approximation occurs because the discrete image can't be perfectly rotationally symmetric.

# Examples (viiib)

$x(n,k) = I\left\{\sqrt{(n/N-1/2)^2+(k/N-1/2)^2} < 0.2\right\}$ with `fftshift`



Rotationally invariant equivalent of a rectangular pulse, and the FT is the rotationally symmetric equivalent of a sync function.

# Radial symmetry

Radially symmetric signal produces radially symmetric DFT

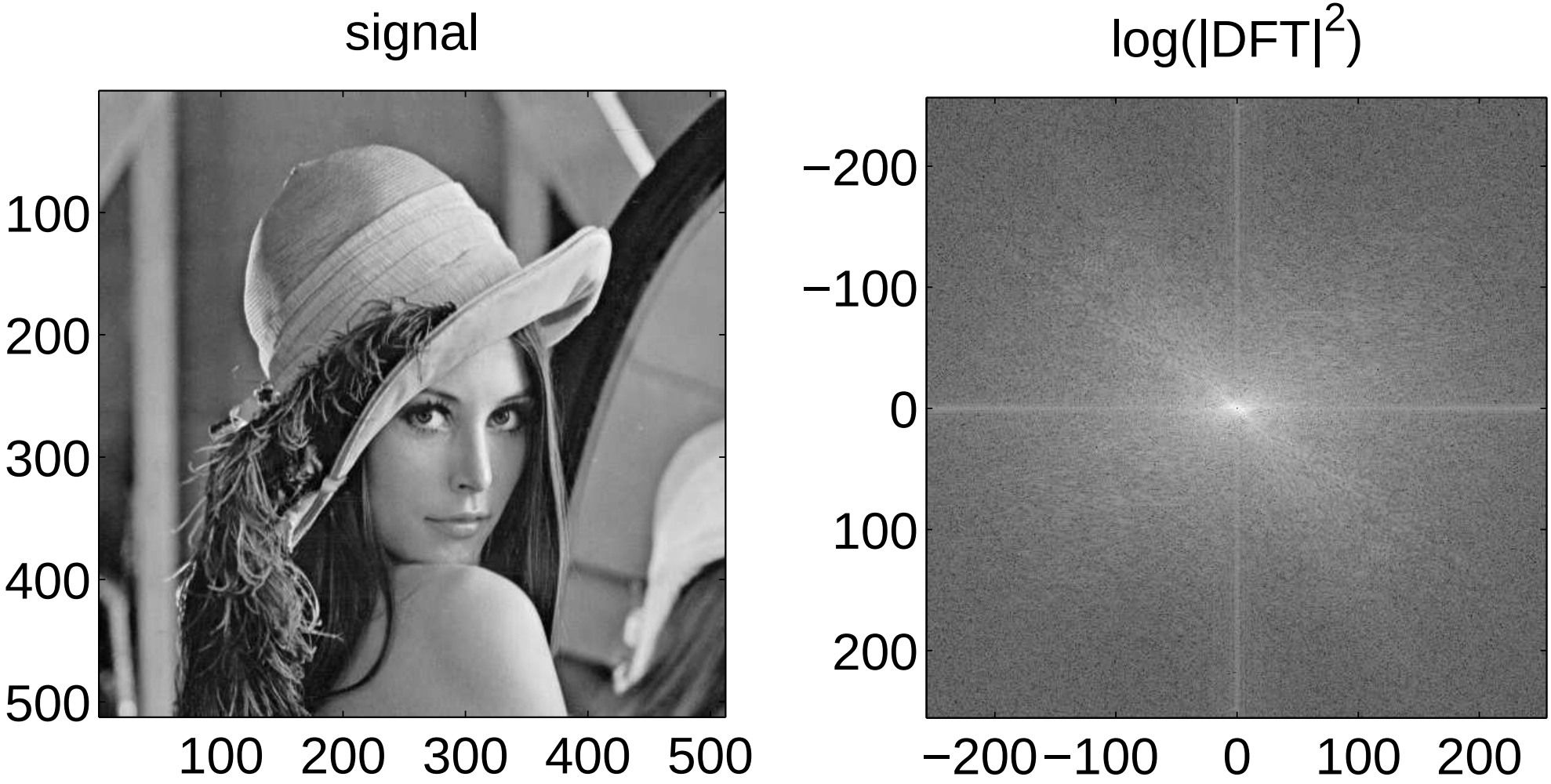
- we know that a rotation in space domain, causes equivalent rotation in frequency domain.
- rotation doesn't change $f(x,y)$, so $F(s,t)$ must also be invariant.
- Remember discretization effects limit radial symmetry.

Given radial symmetry can get Hankel transform:

- useful where the system has radial symmetry
- e.g. optical systems, such as lenses.

# Examples (Lena)

Lena image and power-spectra plotted using `fftshift`



Lena image, see for instance `http://ndevilla.free.fr/lena/`.
Other standard images available at `http://sipi.usc.edu/database/`.

# Aliasing in Images

Just as in 1D signals, when we quantize we introduce noise, and when we sample, we can introduce aliasing. However, aliasing in images (and other higher dimensional signals) can take many forms, and you have probably seen the effect before.

# Aliasing in images

Aliasing in images is similar to that in time signals.

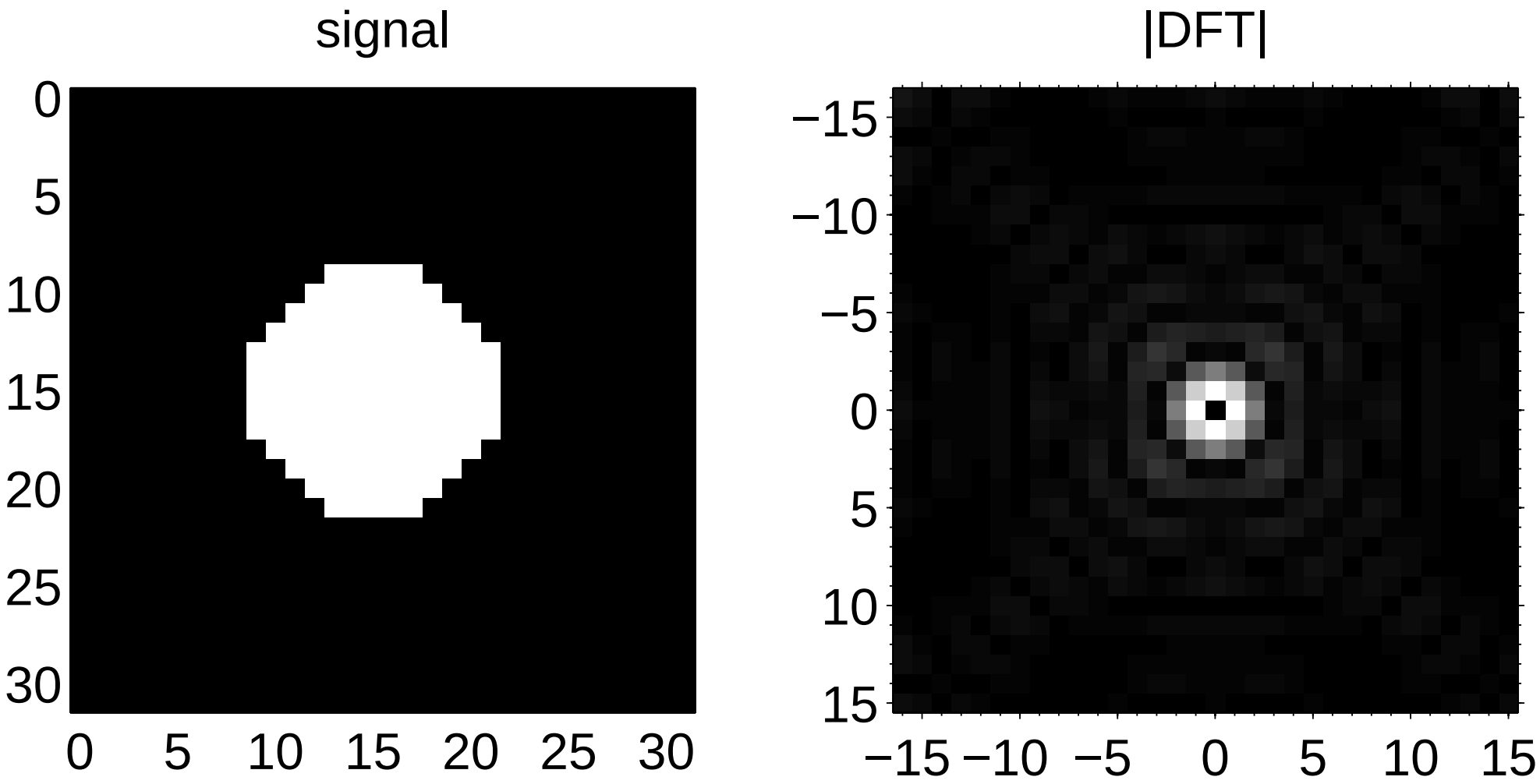
- an image is a sampled spatial field
- cameras average over a small angle for each pixel, so effectively low-pass the image field before sampling.
- CGI generation by sampling of underlying mathematical model.
- produces jagged edges in images "jaggies"
- Moire patterns

Solution is low-pass prefiltering of data (as before).

Other examples of aliasing in higher dimensions

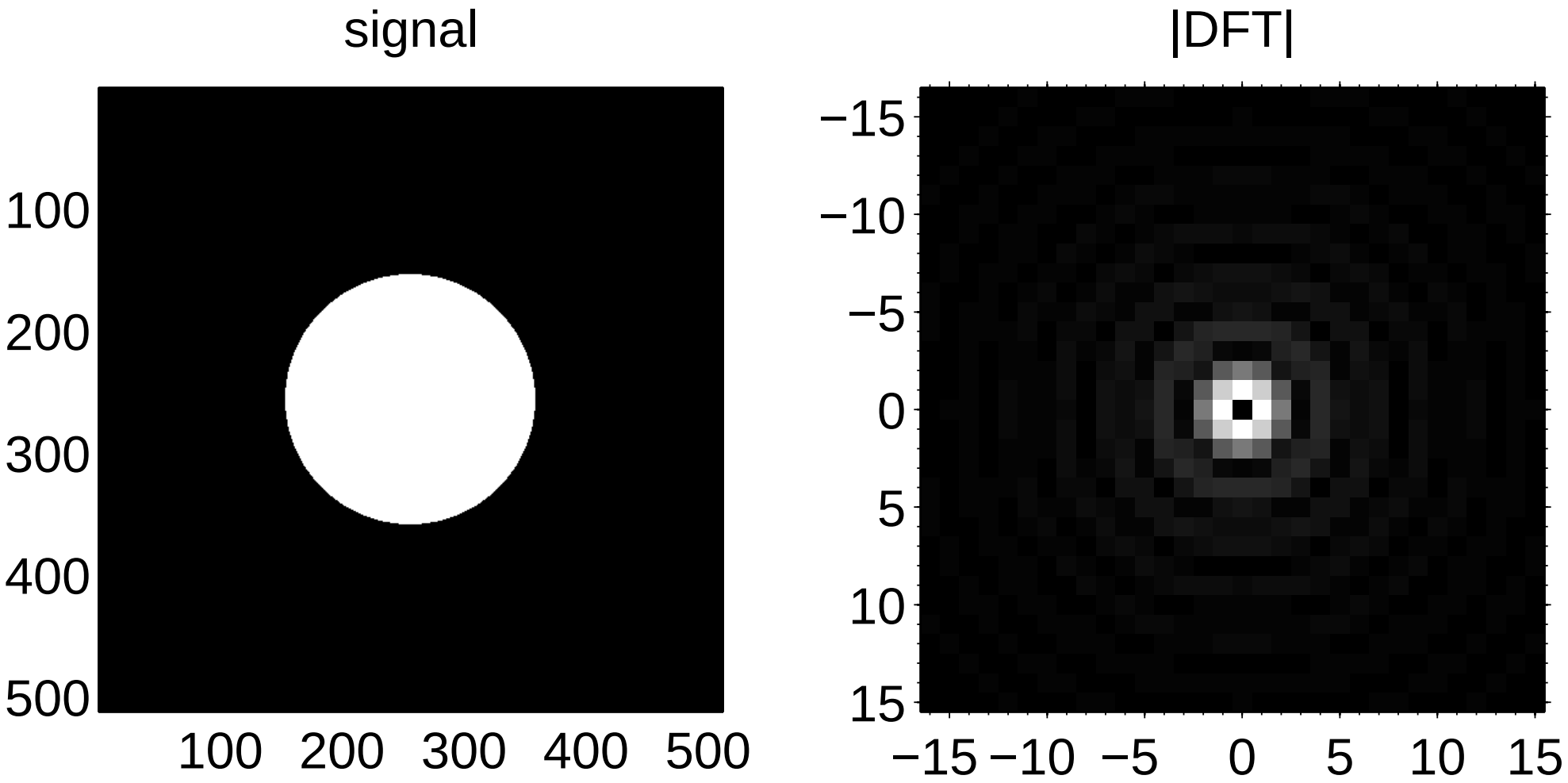
- marching ants in movies
- reverse direction of wheels in movies

# Jaggies



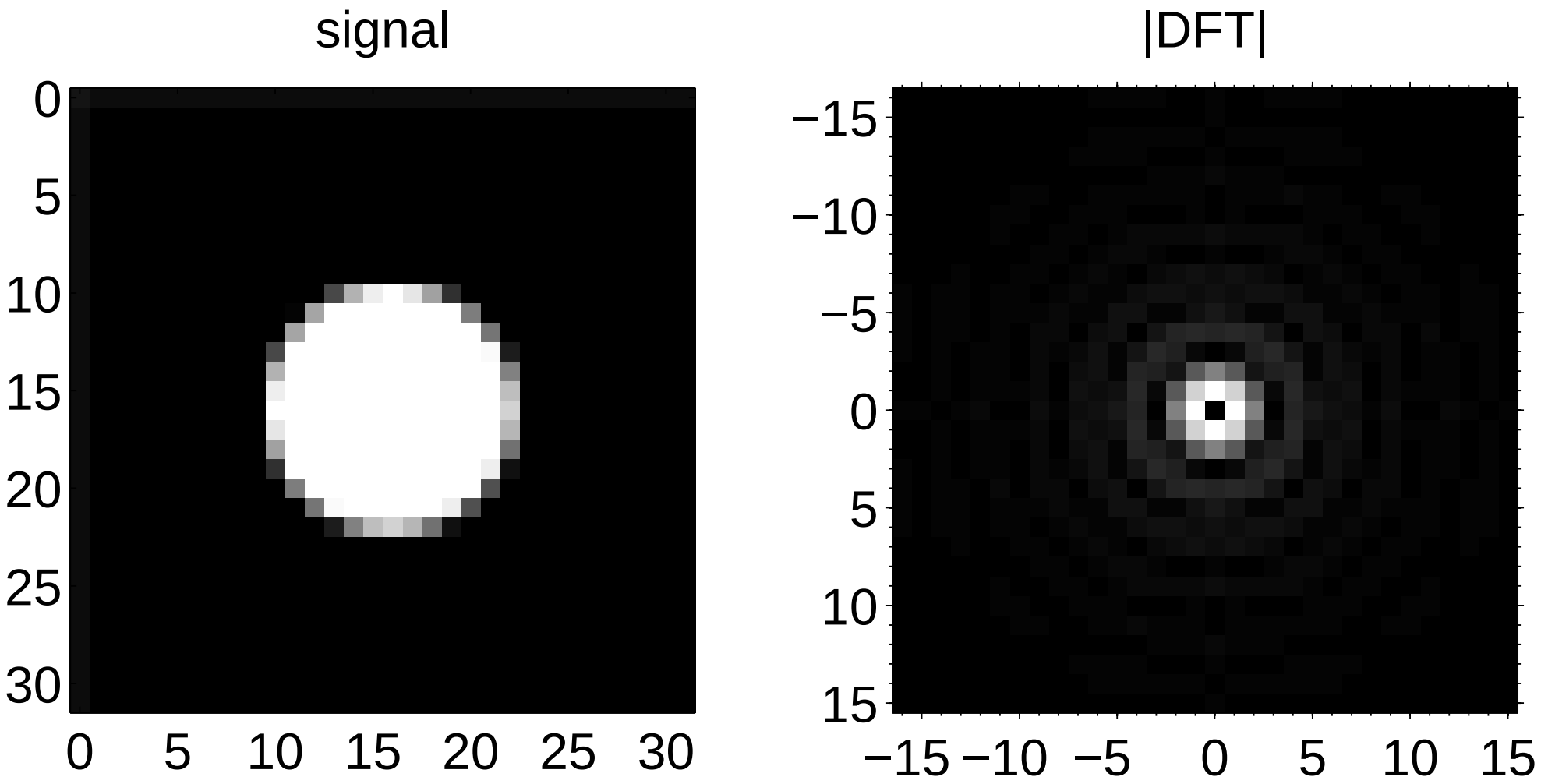
“Jaggies” are the jagged edges you see where a diagonal line in a low resolution image has a jagged edge. It is usually called aliasing in image processing, and it related to frequency aliasing because it is a problem with an insufficient sampling rate.

# Jaggies (reduced by enhanced resoluti



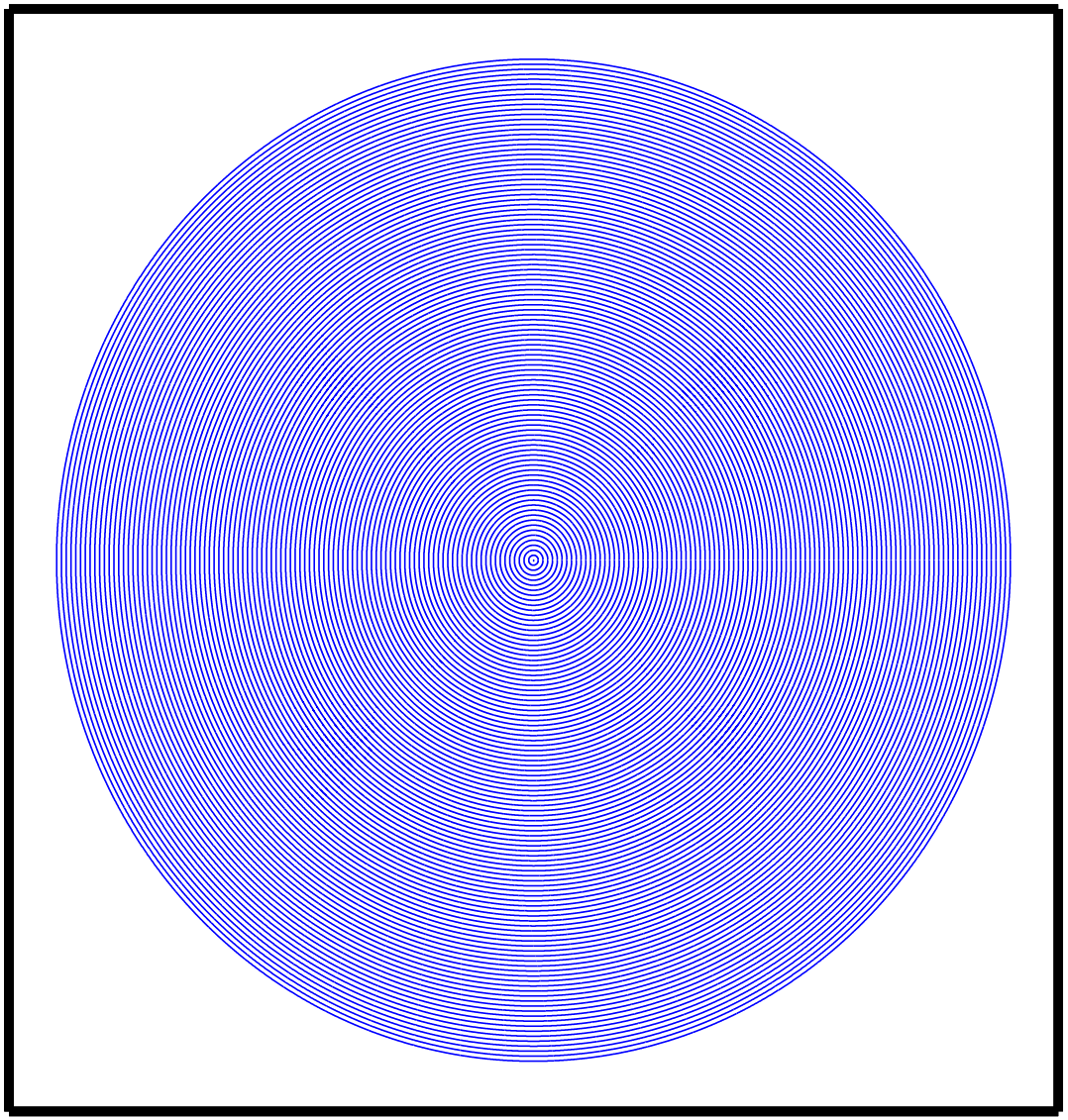
The object being viewed is the same as in the previous slide (a disk) but the enhanced resolution in the image (from $32 \times 32$ to $512 \times 512$) results in a better picture, and a reduction in jaggies. When we look at the DFT (restricting our attention to the same central region of the DFT as before, even though given the higher spatial resolution, we could see a wider range of frequencies now, if we wanted to) we see that the DFT looks pretty similar, but it is now slightly more circularly symmetrical, resulting from the more circular image.

# Jaggies (reduced by pre-filtering)



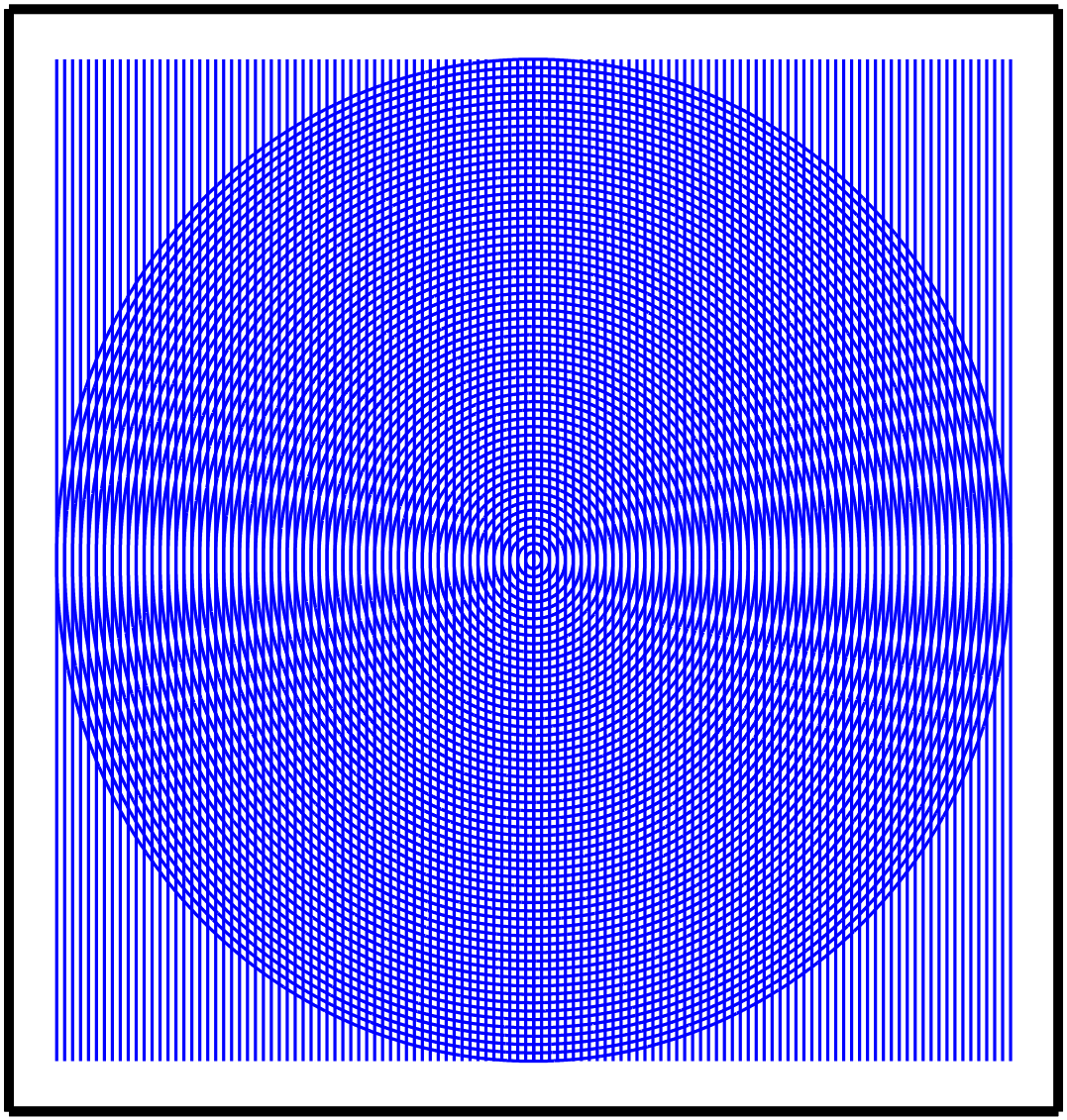
The object being viewed is the same as in the previous slide (a disk) but now when we sample, we don't just take a value (1 if the point is on the disk, and zero otherwise), we take a set of samples at high resolution (see previous image), and average these to get the new value (averaging is a crude low-pass filter).

# Moire patterns



Moire patterns occur when we put many lines together. For instance, in the picture above, only circles are plotted, but we observe other apparent patterns appearing in the image.

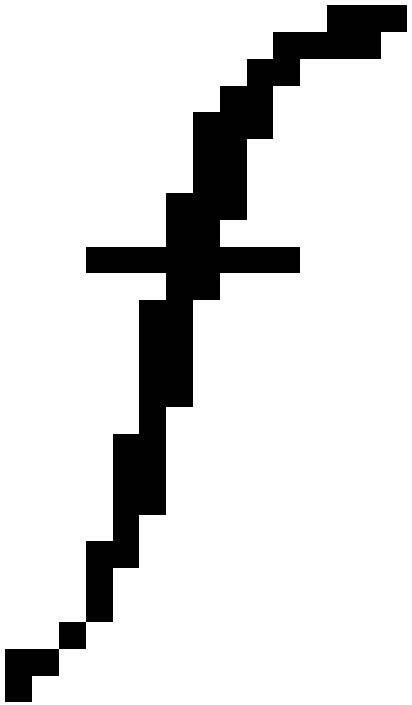
# Moire patterns



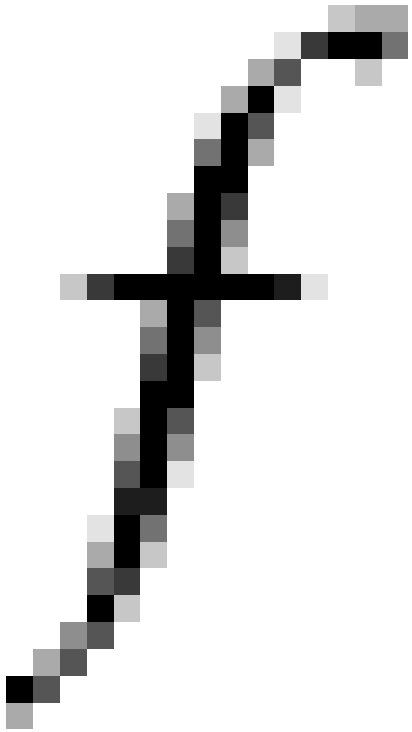
In this example Moire pattern circles are plotted on top of vertical lines. The other patterns are sampling artifacts.

# Anti-aliased fonts

Aliased



Anti-aliased



As before, we avoid aliasing (e.g. jaggies) by low-pass filters. An example occurs in anti-aliased fonts, where we can see the anti-aliased font is blurred. When printed, this produces smoother looking edges on the fonts (all the fonts used in this document are anti-aliased).

# Anti-aliased CGI

Computer-Generated Images (CGI)

- one way to generate is **raytracing**
- generate a ray for each image pixel, and trace its path, including reflections, and refraction.
- computationally expensive (there are faster but less exact methods), so don't want to generate more than one ray per pixel
- but jagged edges move (somewhat randomly), generating marching ants.
- to get **good** results need to **oversample**, and average (e.g. a low-pass operation)

Some references:

http://www.siggraph.org/education/materials/HyperGraph/aliasing/alias2a.htm

# Aliasing: crawling ants

# Resampling applications: video

There are many variations of display for images

| **Display** | **width $\times$ height** | **Depth** | **rate** |
|---|---|---|---|
| VGA monitor | $640 \times 480$ | 4 bit | 60 Hz |
| PAL TV | $720 \times 576$ (625) | - | 50 Hz |
| HDTV 1080i | $1920 \times 1080$ | 8 bit | 50 Hz |
| Plasma TV (typical) | $1024 \times 768$ | 14 bit | variable |
| Film | $\sim 3000 \times \sim 2000$ | $\sim 12$ bit | 24 Hz |
| Workstation | $1920 \times 1200$ | 24 bit | 60 Hz |
| B&W laser printer | $6600 \times 5500$ | 1 bit | - |

We need to be able to convert between these (e.g. if your standard DVD player is hooked up to a high-def plasma screen).

A typical big plasma screen TV

- 106 cm (42 inch) diagonal screen
- 16:9 aspect ratio
- 1,024 x 768 resolution
- 3000:1 Contrast Ratio
  - really talking about dynamic range
  - Q: what quantization would you need to exploit this range? remember RGB
- 14 bit processing

So how do you take a DVD recording a PAL signal, or a HDTV signal and play onto a plasma screen? Obviously we need to resample.

# Resampling applications: printing

Problem of resampling has been around for a long while

- Printers put ink on a page
  - think of this as 1 bit quantization (ink, or no ink)
  - so how can you do a picture, with only 1 bit?


becomes

Obviously you lose a lot of detail when you simply quantise an image to one bit, and print.

# Newsprint

Newspapers came up with solutions many years ago

from the Australian, Dec 11th, 2005

When you look closely at the image you see that it is really made up of many small dots. The size of the dots can vary, as can the density of the dots. This is called “half-toning”.

To see why it works, think of our eye as a DAC. Our eye converts a digital picture, e.g., the newsprint (or a TV picture) from digital (on the page or screen) into an analogue signal in our nervous system. Our eye is acting as the low-pass “interpolation” filter that smooths the samples out to create a nice smooth function (the image we perceive).

There is a simple tradeoff between quantization noise, and the sampling rate. Given a higher sampling rate, we can afford to have more quantization noise, because the “smoothing” effect will smooth over more noise, and thereby reduce it.

# Resampling applications: printing

- many printers have only 1 bit quantization
  - e.g. B&W laser printers
- ink jets (typically) have 3 or 4 color inks
  - but you can't mix them, so need to put together somehow?
- printers typically have better spatial resolution than a monitor
  - e.g. 1200 dpi (dots per inch)
  - compare to a monitor with perhaps 72 dpi
- so we tradeoff sampling vs quantization
  - it works because our eyes (or other sensors) effectively do a low pass before sampling

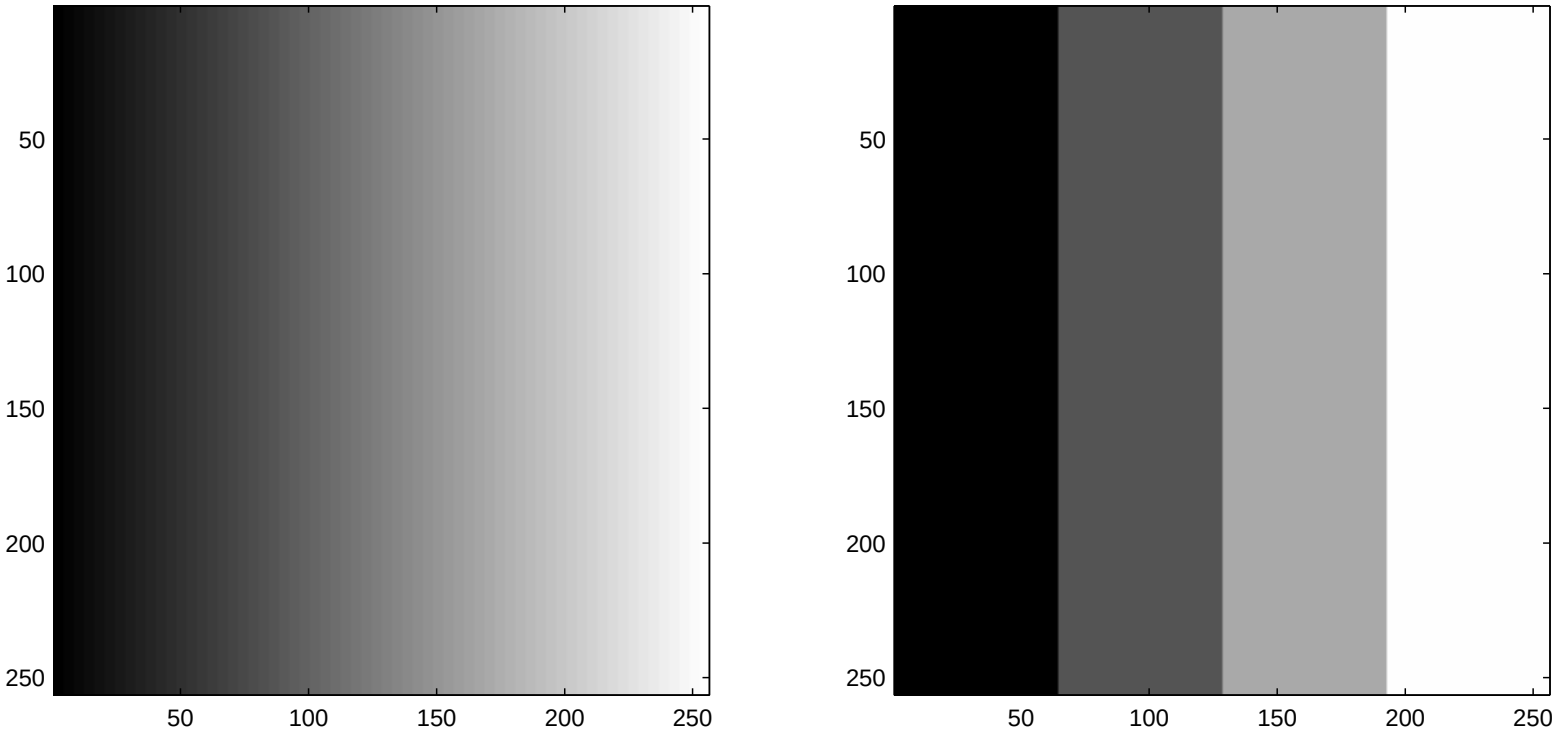
The tradeoff between sampling, and quantization is interesting. Better resolution in one can compensate for a lack in the other.
http://www.mwrf.com/Articles/Print.cfm?Ad=1&ArticleID=10586

The point at which we operate is usually determined by the particular technology costs of the medium we are working in.

However, there are some interesting issues to consider – for instance, we have so far ignored the structure of quantization noise. In fact, quantization noise isn't "white". It can have frequency related effects, and these can be more or less perceptible, so some of the balancing act above is determined by the characteristics of our senses.

# Half-toning

- problem is that we have limited quantization

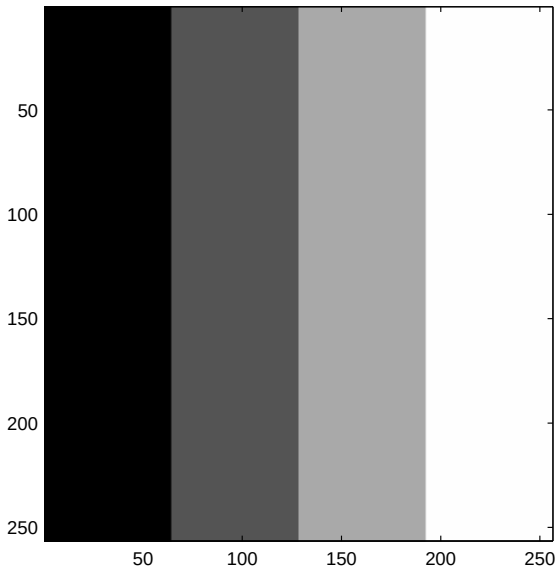


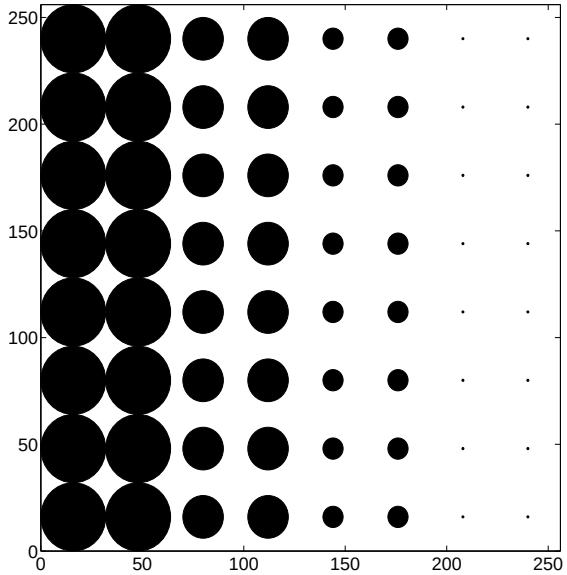

- exchange spatial resolution for quantization
- processing is called half-toning (for printing)

# Halftoning

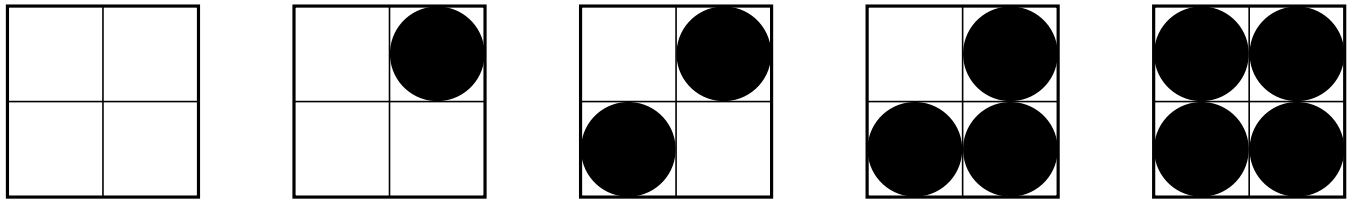
Processing is called half-toning (for printing)

- use dots of varying size




- use patterns of dots



level 1 2 3 4 5

Both techniques exchange spatial resolution for quantization accuracy. e.g. in the pattern of dots, we need twice the spatial resolution to represent 5 levels (instead of 2 levels).

For more information/examples see
http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/dither/sld005.htm
http://www.webopedia.com/TERM/D/dithering.html
http://www.geocities.com/ResearchTriangle/Thinktank/5996/techpaps/introip/manual04.html
http://www.cs.indiana.edu/~dmiguse/Halftone/
http://www.webstyleguide.com/graphics/dither.html
http://www.visgraf.impa.br/Courses/ip00/proj/Dithering1/floyd_steinberg_dithering.html

Matlab code:
http://www.ece.utexas.edu/~bevans/projects/halftoning/toolbox/

# Dithering

Problem with simple approaches above:

- introduce some patterns into the image
- these might be perceptable
- want to add some randomization
- example:

$$P(x,y) = \textsf{round}\left[I(x,y) + \textsf{rand}(x,y)\right]$$

- Example applet
  `http://www.markschulze.net/halftone/`

Its a strange idea. The introduction of random noise can actually improve the perception of an image. The issue is that its easy to compensate for some types of noise, but not others. Its used in audio recording as well as in images to make sure that the underlying quantization noise (floor) is “white”, i.e., doesn’t have patterns.

The approach above is rather simplistic – there are many much more sophisticated ways to introduce randomization.

# Examples of Dithering


1. original image
2. dithered by adding a random value
3. Floyd-Steinberg error diffusion dithering

Obviously, these images aren't as good as possible, because I have to be able to display it on a screen (e.g. monitor) so that you can see the dots. When printing, the dots are much smaller, and the image quality is much better. Compare your printed notes to what is displayed on the screen. In a real application the dots would be smaller than can easily be seen by the naked eye. Our eye acts as a sort of low-pass filter blurring the individual points together to form shading.

# Extreme example: ascii art

- Sample the image down to grayscale with less than 8-bit precision, and then assign a character for each value.
- Ideally character is related to grayscale
- also shape can be used to help define lines.

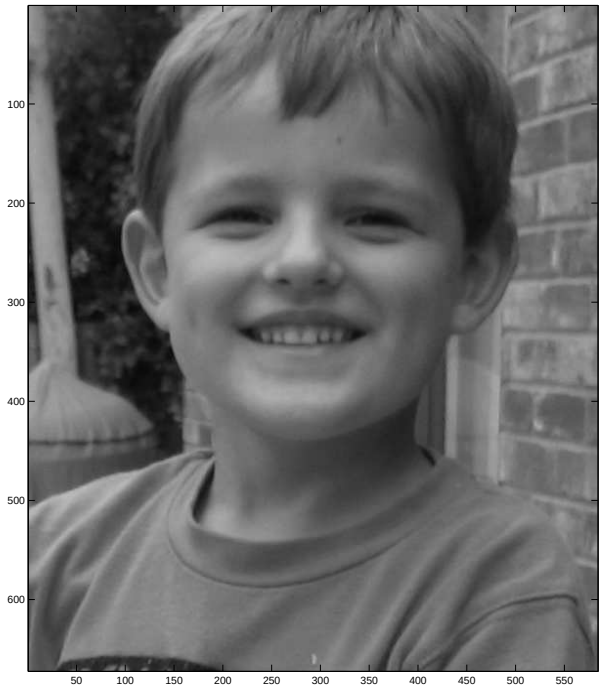
Comments on ascii art
`http://xenia.media.mit.edu/~nelson/courses/mas814/`

Code to make ascii art images
`http://aa-project.sourceforge.net/index.html`

On-line converter
`http://www.text-image.com/convert/`

Examples
`http://www.chris.com/ascii/`

# Extreme example: ascii art




⇒

```
=IZm#mmWWQWQQQW&>+-:::==vvSnXYY1uXXXSX1isilvvSX1><ii|:=|vv
|vZ#mBWWWWWWWWY~:===||indvvnovooonu1o1iv32owoXuI><|i=+=%v}
oomWWWWWWBWQB(<suqX1vvn*+<omqm#XmXXqZXoXqqZ#Bmmoc||||=ii+=
({$WWWWWWWW#nommZ1|li|:::<mWmmm#X21XZo#mBWWWWWWQmoo2>=il|s
;+3WWWWWWWW&m#WBl|||+++||vXTYYY1lIvvXXX#mmWWWWWBBm2I|vvvvv
:=]QWWQWWmWm#BWel||+|||||||+|||iiIvn2XXZ#BWWWWWWWW1nSSovnn
;;<QWQWQWWW#Wm#ei=:;==|||||||ivvIvIII1X#mWWWQWWWEin1****|
>:<$WWQWWmmmWBZ+=;|ivXXZZXv|ivXmZ##ZXwwwX##WWQWWWk><==>=|l
c:=dWWQWQW1v$#>;<vwm#mWBmZ+:-<X#mWWWWmmm####WWQWWZvviiiiiI
C:;3QWWQWEX#SS;:--~+"!!"+:: .=iI*YYYYYYYSX##WWWWWmovvv<nns
h=>)WmWWQQ2evd>::....:_===_=iiuna>;:;==ivo##mWm###1nvsi2no
Q=1=WWWWWWm%il====i%uoX1IX#mZmm#XXXuauoqX###m##XX+==<vvvvs
W>+=3QWWWWWQa;;|vIvSXmasaawXmmmqwwXZXXZ#U###2XXe=:==|i||ll
Xc;=]WWWWWWWWQp|||iI1YVVX11v13XX#UVS1nXZZ##YXXv>=><%ii>+++
?^=+IWBBWBWWQWWoilil|=:===|illvvviilvXZ##m#=n1i=<=|||i=+++
:====*?TV$QWQWWWsivvi;::--=-=-++=inomX#mmB#=vi|:)ss>|isiIi
;=====|=ivd$WWQ#liiInna>||||iiaawm##mWWWmm#=v%|:)S21vXXXZo
sssvaouwwqmm#WWW>{ollvnXXqqqqmm#mWmWWWWBmWE|2o|:=|IIi*l*Yv
nnauXqd###ZVTT*l|}n%ivnnXZ##mmmBWBWWWWBBW#XoX1|;=innovvvv|
XX1Y!""~..iu2}===aB||v2XXX##X##mmBmm###Z#ZZSoonsiiII1vvvi|
+=:.;;==uZY|=:;:<#(-=<1IllIISZ####X#Z#Z#X2onv1li|||||i=<iv
i|==a|nX2<||====<nas>>|i|svuqqqXXZZXXSooo1vno1llvaonli||I1
+:vZ1oY+iv>==++||ilv2XUU###XXXSXXXX21no1vuZ2lnomZSvii|vlll
aomXva=vl+=|+|=|||IIvnnnoooXXXX2onlvnoIumS1lomZYlisvIvsvnS
Sm#mZiv|=+|=++=|+|ilIvvvoXXSnnvn1ivoIvwZ1ivd#XonnoXS2113#v
XZmZ|||=+||+||=+||ivnoXXXnnvvvvl|v1|vZSulvdZZYsuX2nvvilnZh
dZ#oawwwwpwwwwwwwumqXX1IlIvvI|||%i<w#nZoXqmSvoXX11ili|vXX#
```

# Applications of Transforms in 2D

There are many examples of applications for 2D transforms. We examine briefly JPEG image compression and digital watermarks.

# Application: JPEG compression

Some references:

http://www.faqs.org/faqs/jpeg-faq/
http://www.photo.net/learn/jpeg/
http://www.cs.cf.ac.uk/Dave/Multimedia/node234.html
http://netghost.narod.ru/gff/graphics/book/ch09_06.htm
http://cobweb.ecn.purdue.edu/~ace/jpeg-tut/jpegtut1.html
http://www.imaging.org/resources/jpegtutorial/index.cfm
http://delivery.acm.org/10.1145/330000/327649/a2-hankerson.html?key1=327649&key2=3302073511&coll=portal&dl=ACM&CFID=25660063&CFTOKEN=26680067

# Application: JPEG compression

Example stats

| **compression** | **size (kb)** |
|---|---|
| no compression | $2304 = 1024 \times 768 \times 24/(8 \times 1024)$ |
| JPEG quality=0.75 | 62 |
| JPEG quality=0.50 | 37 |
| JPEG quality=0.25 | 24 |
| JPEG quality=0.10 | 13 |
| JPEG quality=0.05 | 9 |

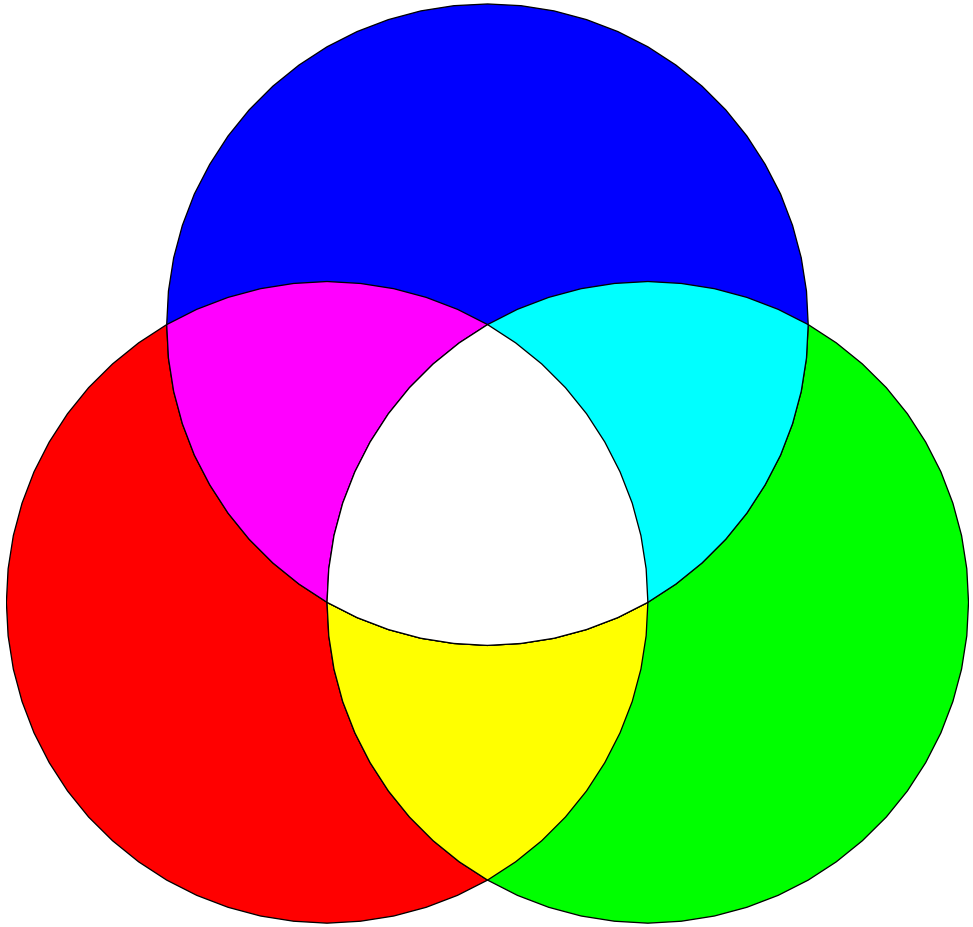
# JPEG algorithm

Steps:

- color transform RGB to YIQ, and downsample I, Q
- divide image into blocks of 8x8
- For each 8x8 block
  - DCT
  - quantize
  - encode quantized bits

# Color in images

- Primary colors = Red, Green, Blue (RGB)
- combinations of these give colors



| Color | Hexadecimal |
|---|---|
| aqua | #00ffff |
| gray | #808080 |
| green | #008000 |
| lime | #00ff00 |
| maroon | #800000 |
| navy | #000080 |
| olive | #808000 |
| purple | #800080 |
| red | #ff0000 |
| silver | #c0c0c0 |
| teal | #008080 |
| white | #ffffff |
| yellow | #ffff00 |
| black | #000000 |
| blue | #0000ff |
| fuchsia | #ff00ff |

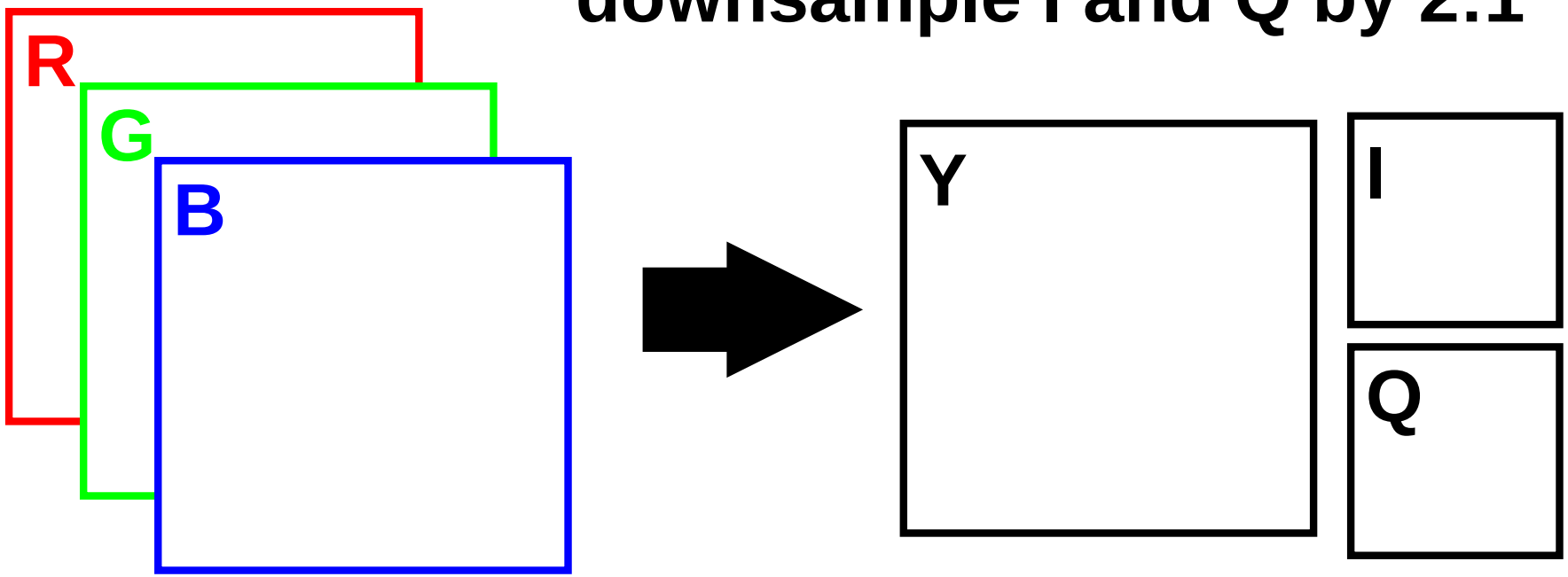
Figure shows just a few examples of mixing colors to get new colors. For other examples see, e.g.

http://en.wikipedia.org/wiki/Web_colors
http://web.njit.edu/~kevin/rgb.txt.html
http://www.is.kiruna.se/~cjo/d2i/COLOR.RGB.html
http://www.kenjikojima.com/java/RGBHexConverter2.html

Often each color is quantized with 8 bits, and so takes values from $0 - 255$, and each can be represented by 2 hexadecimal digits, so one might write a color like #A62A2A, e.g. HTML colors shown in the table above.

# JPEG compression (color transform)

**convert RGB to YIQ and downsample I and Q by 2:1**




We can downsample I and Q because our eyes are less sensitive to these components.

Some color space transformations:

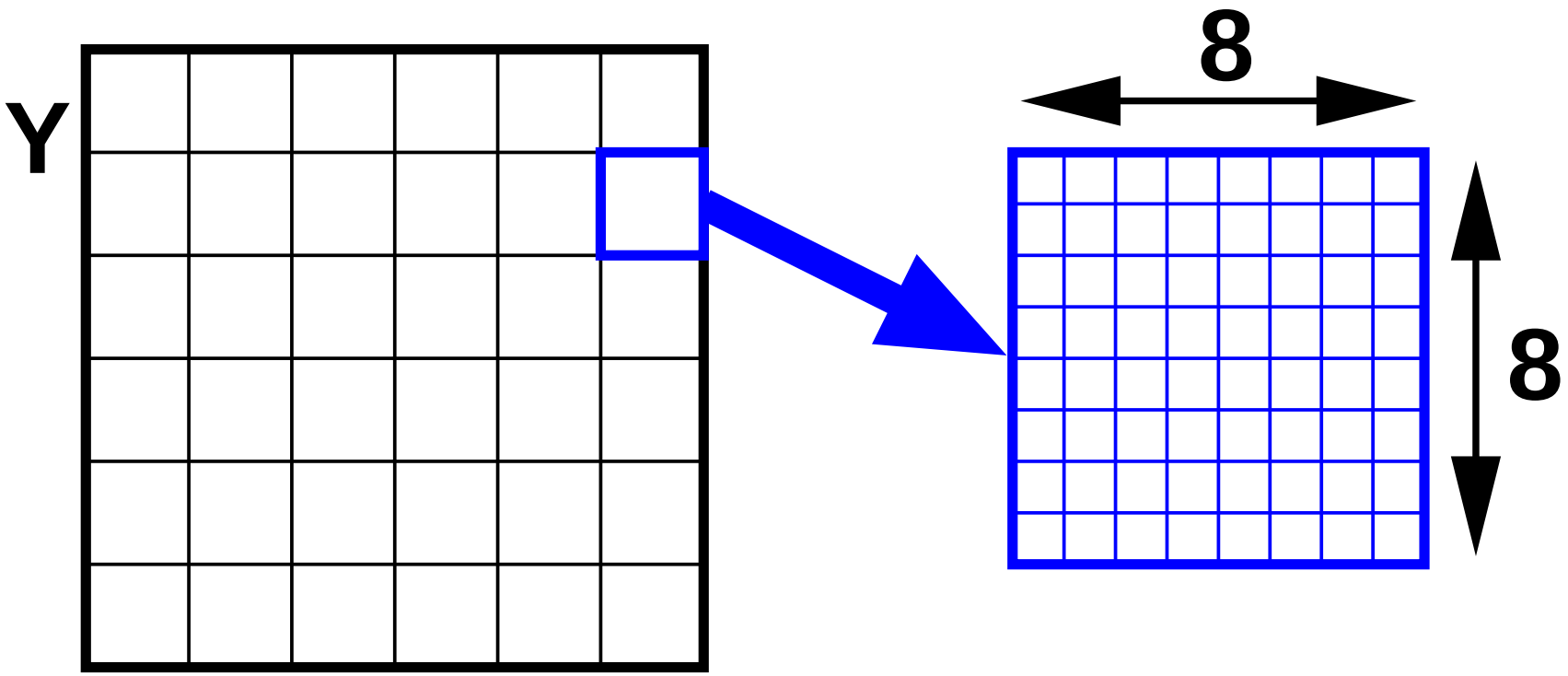
$R, G, B \in [0, 1]$

$Y \in [0, 1]$, $I \in [-0.595716, 0.595716]$ and $V \in [-0.522591, 0.522591]$

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595716 & -0.274453 & -0.321263 \\ 0.211456 & -0.522591 & 0.311135 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Other forms of color transform, YUV, YCbCr (used in JPEG2000), and so on

# JPEG compression (blocks)




**break image into 8x8 blocks**

# The Discrete Cosine Transform (DCT)

For an $N_1 \times N_2$ image $A$, the DCT is defined as

$$B(k_1,k_2) = 4 \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} A(i,j) \cos\left[\pi \frac{k_1}{N_1}\left(i+\frac{1}{2}\right)\right] \cos\left[\pi \frac{k_2}{N_2}\left(j+\frac{1}{2}\right)\right]$$

Real-even DFT of half-shifted input

As before there are other possible definitions (e.g. see http://en.wikipedia.org/wiki/Discrete_cosine_transform).

The input image A(i,j) is N2 pixels wide by N1 pixels high. For JPEG, the input is an 8 by 8 array of integers. This array contains each pixel's gray scale level with levels typically from 0 to 255.

B(k1,k2) is the DCT coefficient in row k1 and column k2. The output contains integers, which can range from -1024 to 1023.

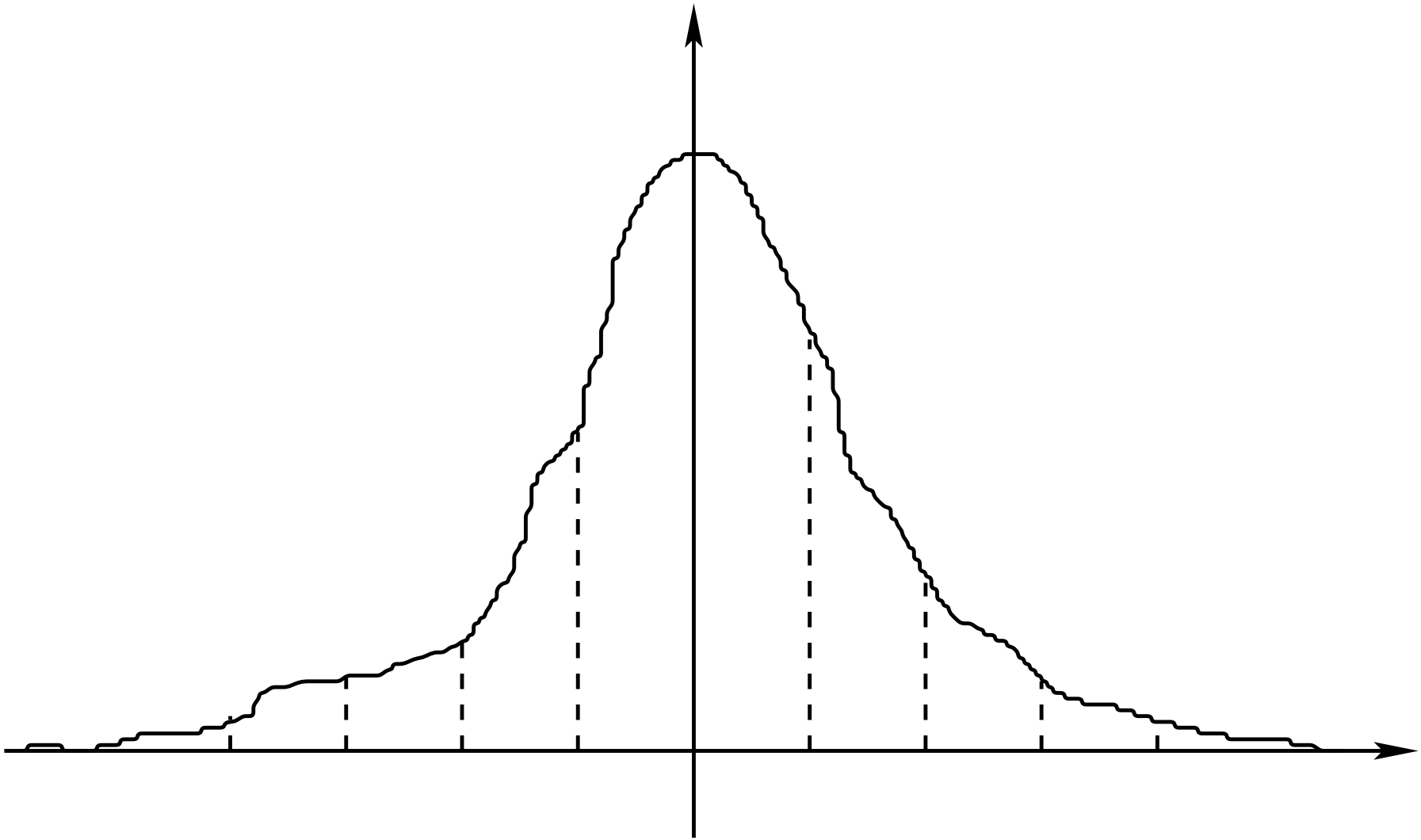
All DCT multiplications are real. This lowers the number of required multiplications, as compared to the discrete Fourier transform. Fast DCTs exist (or it can be performed using FFTs, but this negates the above advantage, though sometimes it might be worth it to take advantage of highly optimized code.).

# JPEG compression – quantization

- quantization refers to setting the values to discrete values.
- if a small number of values are used they can be encoded in a small number of bits (e.g. 4 values, in 2 bits).
- tradeoff between quality and compression.
- uniform compression applies the same quantization across all DCT coefficients.
- high-frequencies in images aren't as visible to naked eye
- hence quantize higher frequencies more, with minimal loss in quality

# Quantization

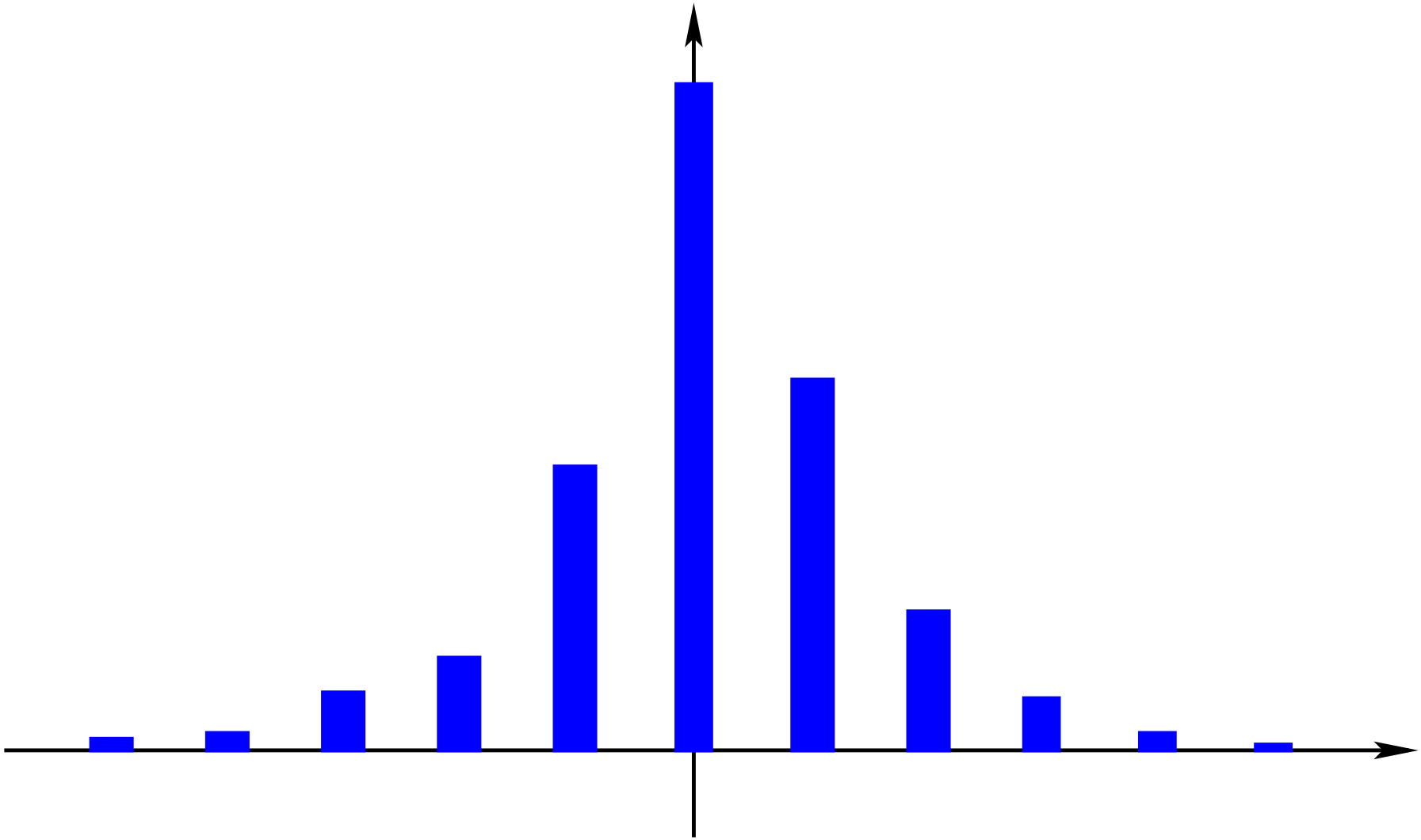
We start with some continuous distribution



Assume the pixels of the image take a range of values, and the plot above gives us a histogram of those values.

# Quantization

Then quantize the distribution into a number of levels



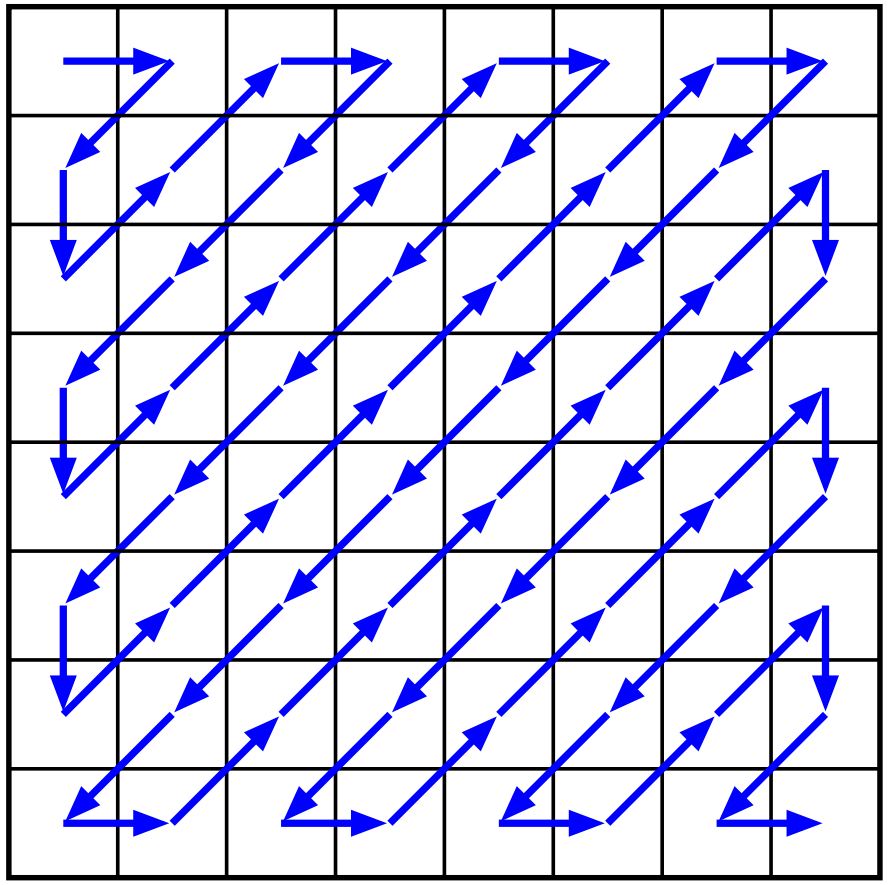
Remember, quantization introduces noise (reduced by roughly 6dB per bit). Actually, we quantize something that has already been quantized once. Typical image values are 8 bits (0-255). Imagine quantizing to 4 bits. This would introduce approximately 24dB of quantization noise, but we do this in the frequency domain. Quantizing frequency $f$ introduces correlated noise into the image, which is not very visible for high-frequencies.

# Quantization matrix

- the **quantization matrix** is an $8 \times 8$ matrix of step sizes for quantization of the corresponding element of the DCT.
- quantize each element by $\lfloor F[k_1, k_2]/q(k_1, k_2) \rfloor$ where $q(k_1, k_2)$ is the quantization matrix.
- top left are lower frequencies, so smaller $q$ values, increasing towards bottom right.
- typically, many values turn out to be zero: this is good for the next stage.

# JPEG compression - Encoding

The levels in the quantized block are encoded sequentially using standard coding techniques, such as Huffman, or Arithmetic encoding, and Run Length Encoding (RLE). The order is important, because the quantization will leave many zeros, mostly in the lower right quadrant. RLE can effectively encode these groups of zeros.



We won't cover encoding techniques in this course: see "Coding and Cryptology" for more details.

# JPEG compression properties

- JPEG compression is lossy (looses information)
- one can't get back to the original just from the compressed information
- loss vs quality is tunable by changing the quantization tables.
- somewhat adaptive (through small blocks), but not very.
- low quality introduces observable artifacts

# Application: JPEG compression

Quality = 0.05

# Steganography

**Steganography:** (covered writing) The art and science of hiding information by embedding messages within other, seemingly harmless messages.

- has often been used to code information in text (see following example)
- more recently, used to encode info. in other forms of data
  - images
  - audio
- coding can be done using least significant bits, so it appears as noise
- we can be even cleverer!

References:
http://www.jjtc.com/stegdoc/steg1995.html

# Steganography

The German Embassy in Washington, DC, sent these messages in telegrams to their headquarters in Berlin during World War I (Kahn, 1996).

PRESIDENT'S EMBARGO RULING SHOULD HAVE IMMEDIATE NOTICE. GRAVE SITUATION AFFECTING INTERNATIONAL LAW. STATEMENT FORESHADOWS RUIN OF MANY NEUTRALS. YELLOW JOURNALS UNIFYING NATIONAL EXCITEMENT IMMENSELY.

APPARENTLY NEUTRAL'S PROTEST IS THOROUGHLY DISCOUNTED AND IGNORED. ISMAN HARD HIT. BLOCKADE ISSUE AFFECTS PRETEXT FOR EMBARGO ON BYPRODUCTS, EJECTING SUETS AND VEGETABLE OILS.

# Steganography

Reading the first character of every word in the first message or the second character of every word in the second message.

PRESIDENT'S EMBARGO RULING SHOULD HAVE IMMEDIATE NOTICE. GRAVE SITUATION AFFECTING INTERNATIONAL LAW. STATEMENT FORESHADOWS RUIN OF MANY NEUTRALS. YELLOW JOURNALS UNIFYING NATIONAL EXCITEMENT IMMENSELY.

APPARENTLY NEUTRAL'S PROTEST IS THOROUGHLY DISCOUNTED AND IGNORED. ISMAN HARD HIT. BLOCKADE ISSUE AFFECTS PRETEXT FOR EMBARGO ON BYPRODUCTS, EJECTING SUETS AND VEGETABLE OILS.

# Steganography

Reading the first character of every word in the first message or the second character of every word in the second message will yield the following hidden text:

PERSHING SAILS FROM N.Y. JUNE 1

# Applications: digital watermarks

Watermarks are a particular case: we want to add to an image user data that is

- recoverable (given the key)
- hard to detect (without the key)
- hard to get rid of

allows one to find copyright violations. e.g.

- finding (and proving) that an image on a web page was illegally copied from its owner.

# Watermarks

VERY non-trivial to get right (security is always hard).
It must resist natural transformations of the image, e.g.

- image cropping
- recompression
- changing file types
- rotation

Plus be hard for an opponent to

- remove deliberately.
- detect and identify
- forge a watermark

Least significant bits would be lost in something as simple as compression!

# Watermarks in the frequency domain

- In JPEG compression, we found that some values could be quantized with minimal visual impact.
- similarly, we could deliberately change the values a little with minimal impact on the image quality
- change low frequency components, as they are less quantized during compression, and less likely to be lost.
- apply to whole image so that cropping doesn't cause a problem
- use crypto techniques on the information to make it hard to access/detect

Some references:


- Berghel, H. (1998). “Digital watermarking makes it mark”, netWorker: The craft of network computing, 2(4), 30-39.
- Cox, I., Miller, M., and Bloom, J. (2000, March 27 -29). “Watermarking applications and their properties”, Proceedings of the international conference on information technology: Coding and computing, Las Vegas, Nevada.
- Dittmann, J., Mukherjee, A., and Steinebach, M. (2000, March 27 - 29). “Media-independent watermarking classification and the need for combining digital video and audio watermarking for media authentication”, Proceedings of the international conference on information technology: Coding and computing, Las Vegas, Nevada.
- Dittmann, J., Stabenau, M., and Steinmetz, R. (1998, September 13 - 16). “Robust MPEG video watermarking technologies”, Proceedings of the 6th ACM international conference on multimedia, Bristol, United Kingdom.
- Memon, N., and Wong, P. W. (1998). “Protecting digital media content”, Communications of the ACM, 41(7), 35-43.

# Applications: digital watermarks

A somewhat easier problem is tamper-proof watermark

- put some mark onto the image such that transformation of the image in any way will damage the mark.
- then if the image has been altered, we can find out
- e.g. for use in chain of evidence, to prove that a digital image was not altered.
- again we can hide this information in the DCT coefficients.