Image features: Histograms, Aliasing, Filters, Orientation and HOG

D.A. Forsyth

Simple color features

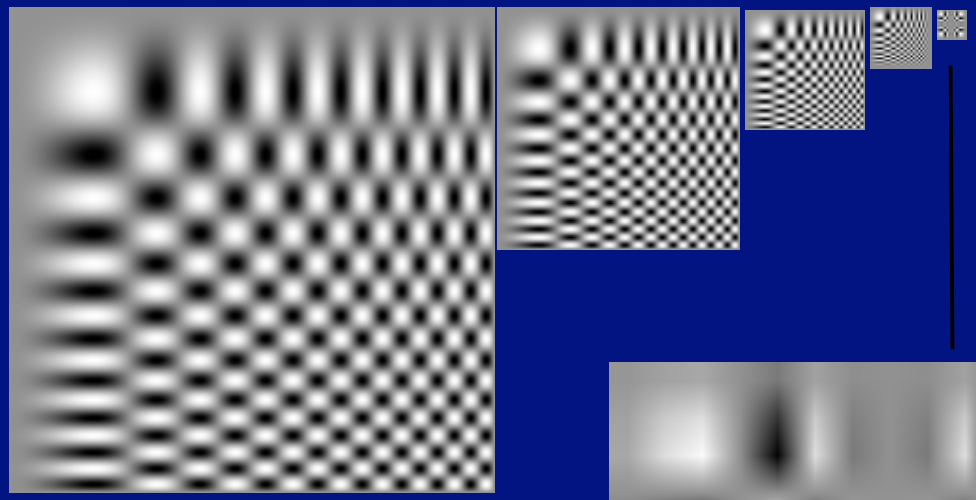
- Histogram of image colors in a window
- Opponent color representations
  - R-G
  - B-Y=B-(R+G)/2
  - Intensity=(R+G+B)/3
- Percentage of blue pixels
- Blue pixel map

```
cd('~/Current/Courses/CS-498-DAF-PS/MatlabCode/ImageFeatures');
pool=imread('mypool.jpg');
bush=imread('bush.jpg');
lawn=imread('lawn.jpg');
%%
rspool=double(reshape(pool, size(pool, 1)*size(pool, 2), 3));
rsbush=double(reshape(bush, size(bush, 1)*size(bush, 2), 3));
rslawn=double(reshape(lawn, size(lawn, 1)*size(lawn, 2), 3));
figure(1);
plot3(rspool(:, 1), rspool(:, 2), rspool(:, 3), 'r*');
hold on
plot3(rsbush(:, 1), rsbush(:, 2), rsbush(:, 3), 'g*');
plot3(rslawn(:, 1), rslawn(:, 2), rslawn(:, 3), 'b*');
%
% notice all the blue pixels
%
%%
% some other color representations
rgpool=reshape(double(pool(:, :, 1))-double(pool(:, :, 2)), size(pool, 1),
size(pool, 2));
bypool=reshape(double(pool(:, :, 3))-(double(pool(:, :, 2))+double(pool
(:, :, 1)))/2,size(pool, 1), size(pool, 2));
intpool=reshape(double(sum(pool, 3)), size(pool, 1), size(pool, 2));
figure(5); imshow([rgpool, bypool, intpool]);
rgs=max(max(abs(rgpool)));
bys=max(max(abs(bypool)));
ints=max(max(abs(intpool)));
figure(6); imshow([(rgpool+rgs)/(2*rgs), (bypool+bys)/(2*bys), intpool/
ints]);
%%
rglawn=reshape(double(lawn(:, :, 1))-double(lawn(:, :, 2)), size(lawn, 1),
size(lawn, 2));
bylawn=reshape(double(lawn(:, :, 3))-(double(lawn(:, :, 2))+double(lawn
(:, :, 1)))/2,size(lawn, 1), size(lawn, 2));
intlawn=reshape(double(sum(lawn, 3)), size(lawn, 1), size(lawn, 2));
figure(2); imshow([rglawn, bylawn, intlawn]);
rgs=max(max(abs(rglawn)));
bys=max(max(abs(bylawn)));
ints=max(max(abs(intlawn)));
figure(3); imshow([(rglawn+rgs)/(2*rgs), (bylawn+bys)/(2*bys), intlawn/
ints]);
```

Scaled representations

- Represent one image with many different resolutions
- Why?
  - find bigger, smaller swimming pools

Carelessness causes aliasing

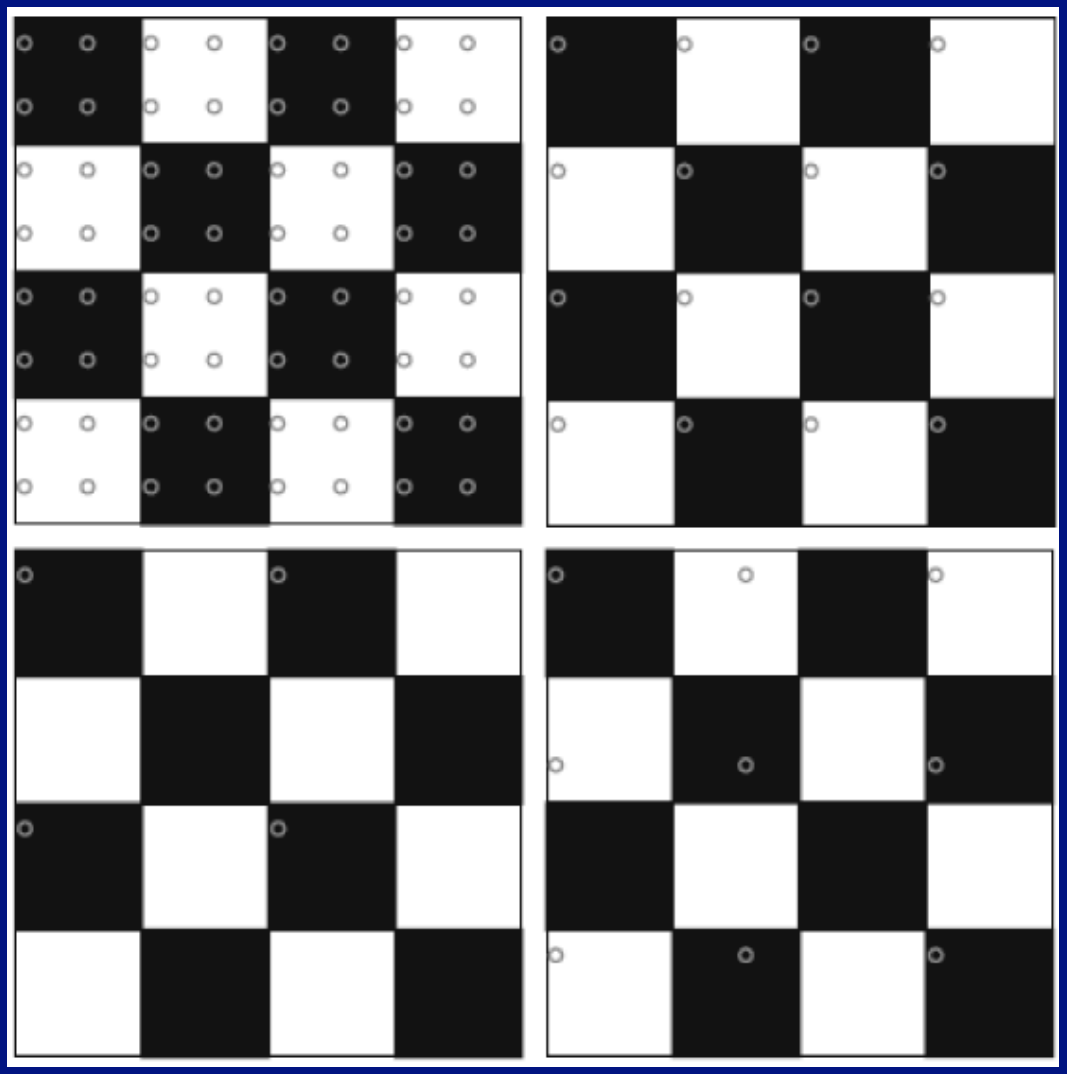


Obtained pyramid of images by subsampling

Matlab slide: subsampling

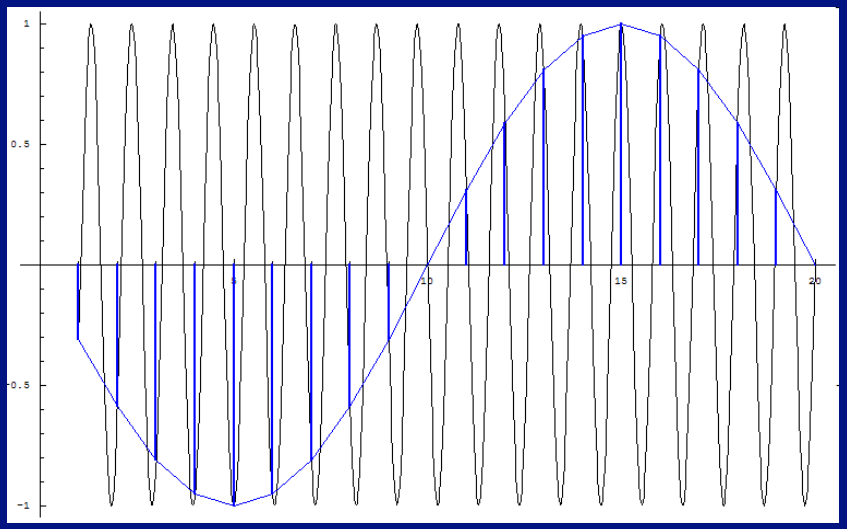
```
cd('~/Current/Courses/CS-498-DAF-PS/MatlabCode/ImageFeatures');
wind=imread('satpic.jpg');
windsmall=wind(1:16:size(wind, 1), 1:16:size(wind, 2), :);
winds2=imresize(wind, 1/16);
figure(1); clf; hold off; imshow(windsmall);
figure(2); clf; hold off; imshow(winds2);
```

Aliasing and fast changing signals



More aliasing examples

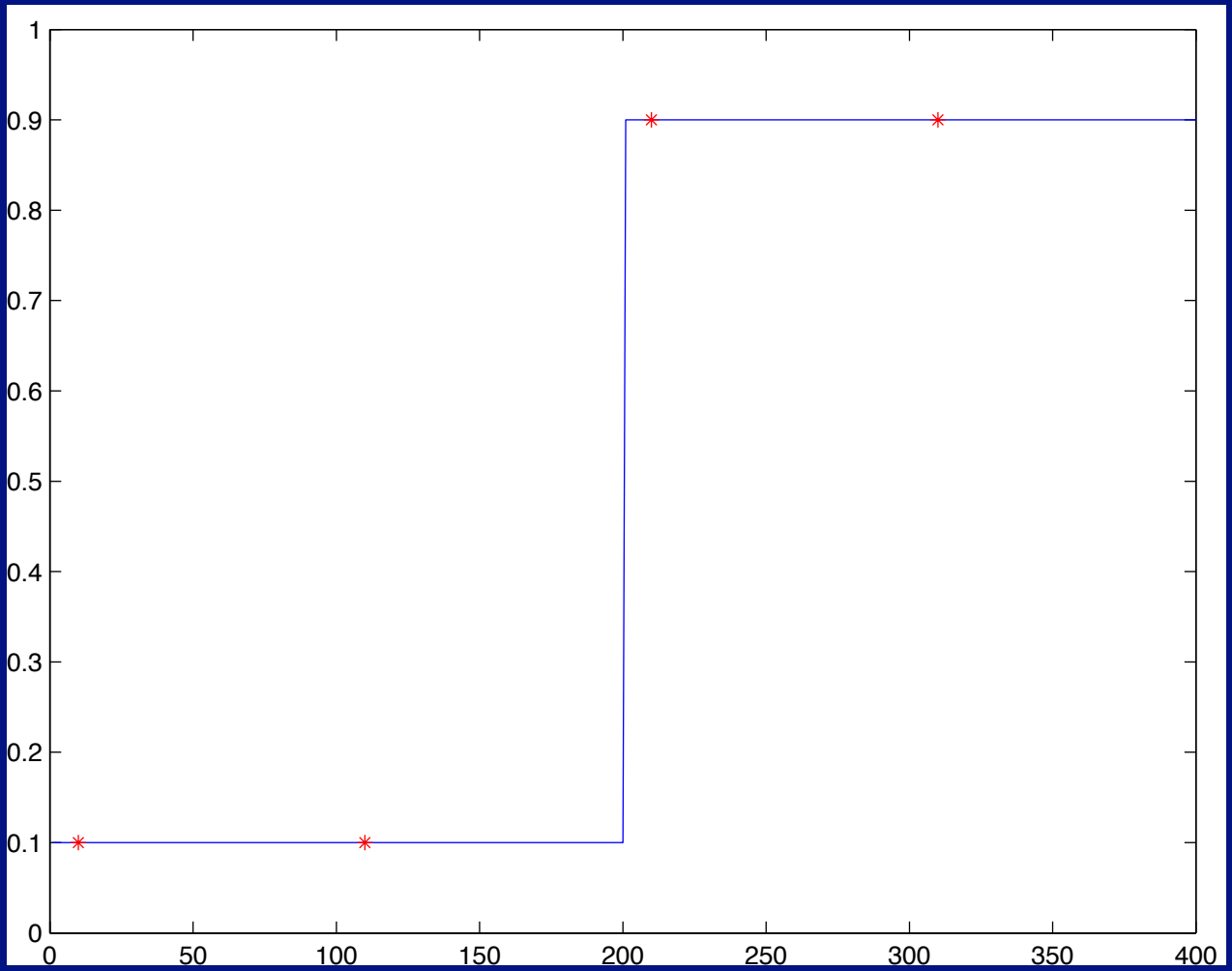
- Undersampled sine wave ->



- Color shimmering on striped shirts on TV
- Wheels going backwards in movies
  - temporal aliasing

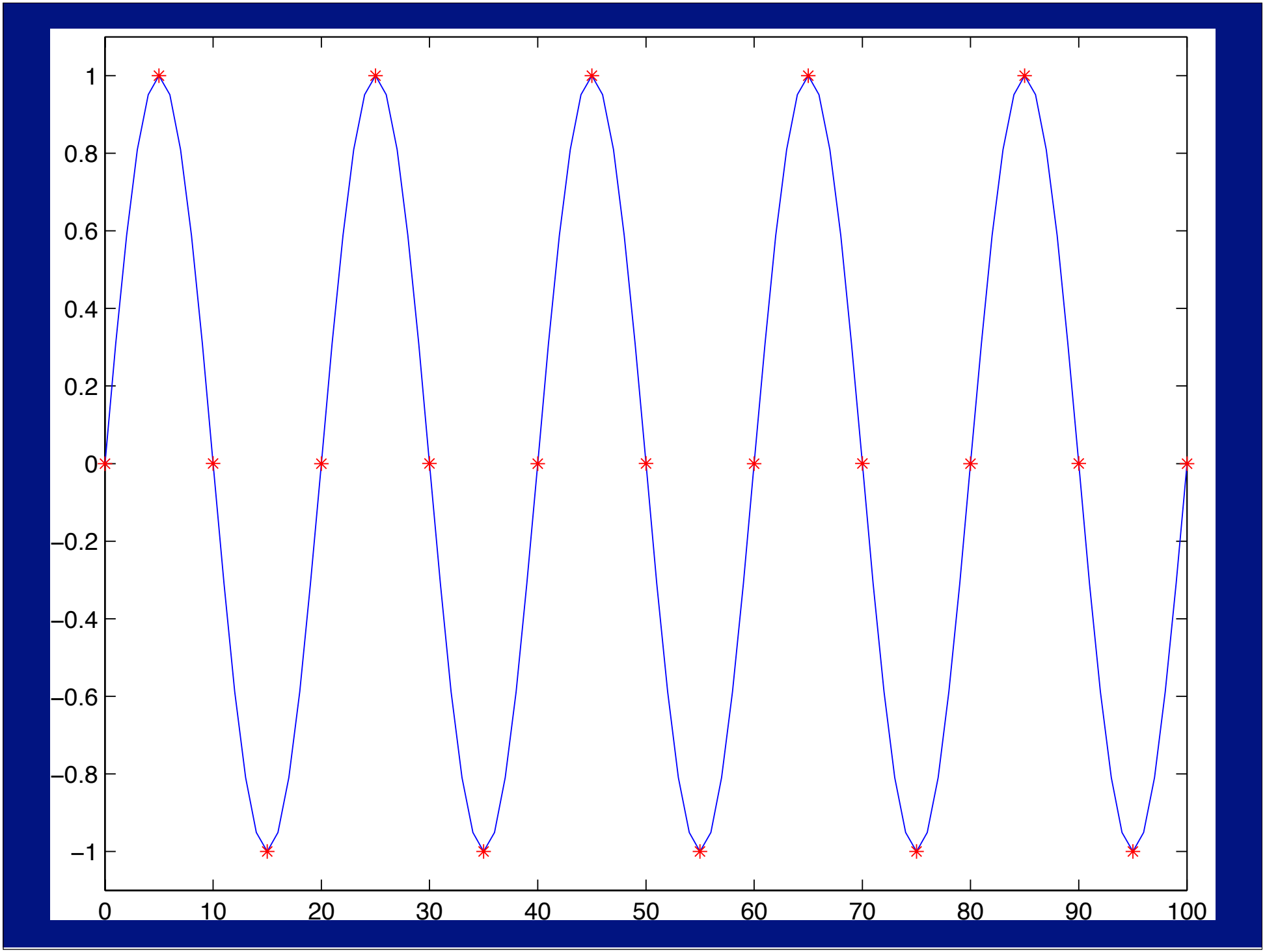
Another aliasing example

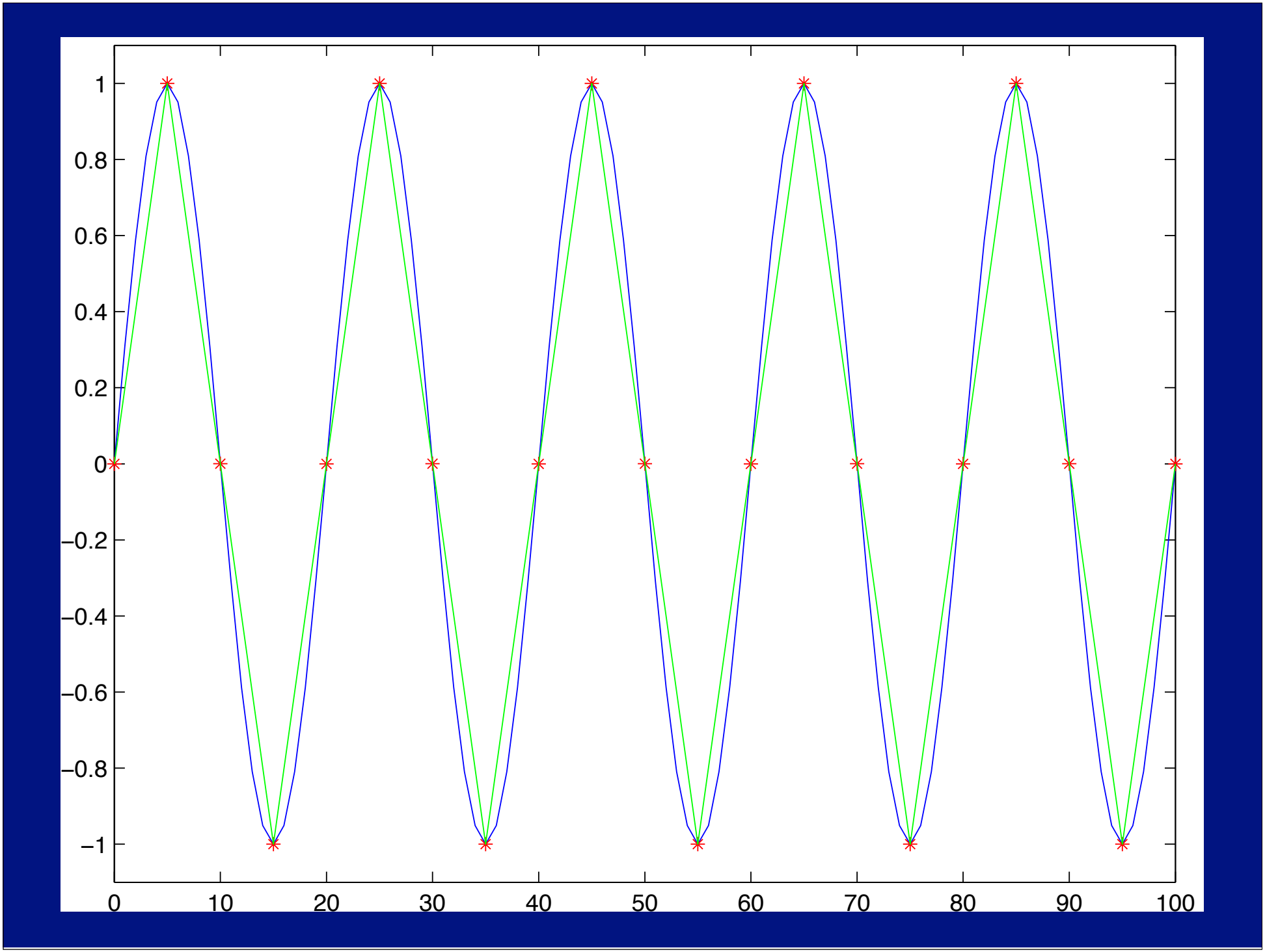
- location of a sharp change is known poorly

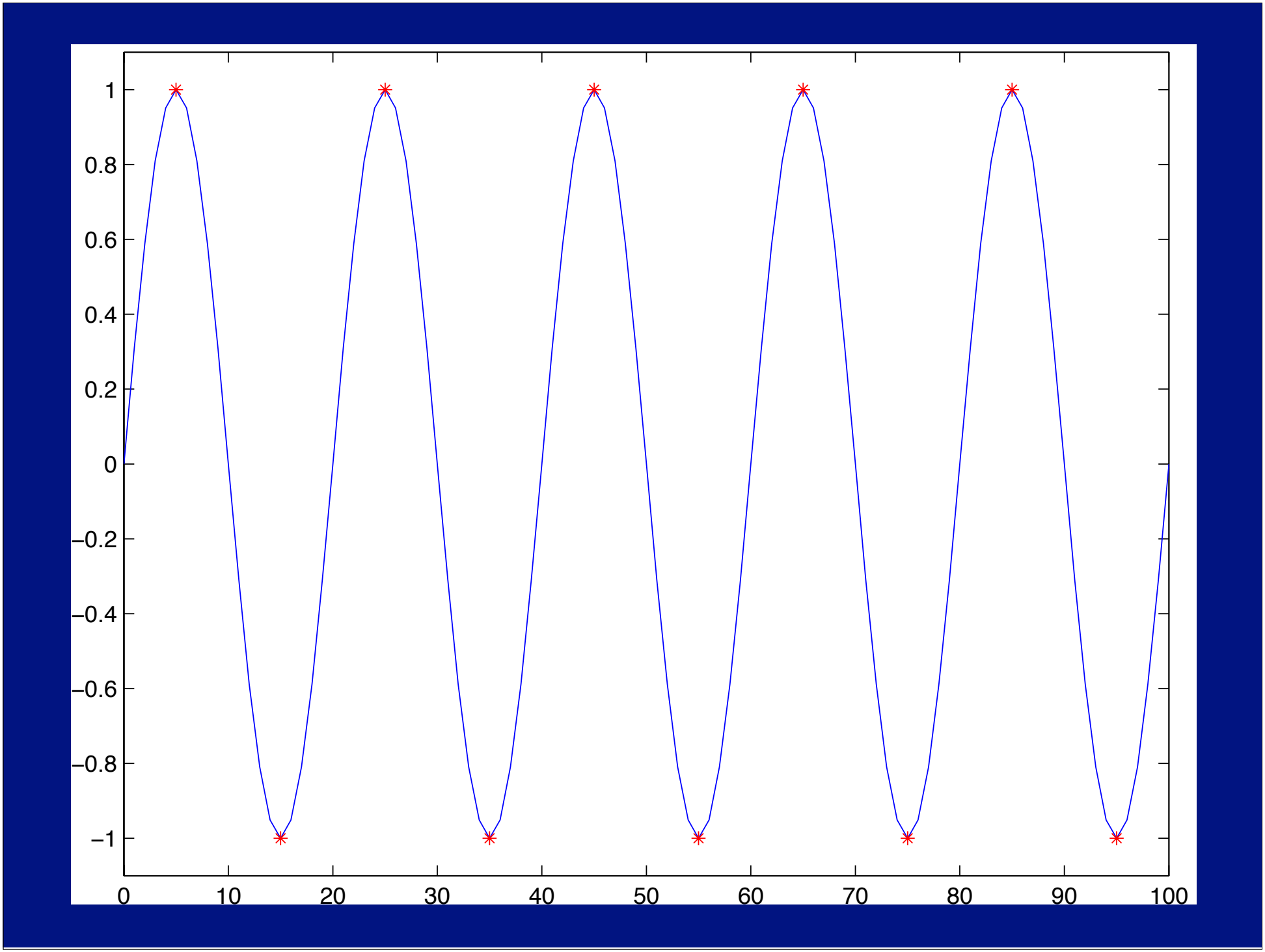


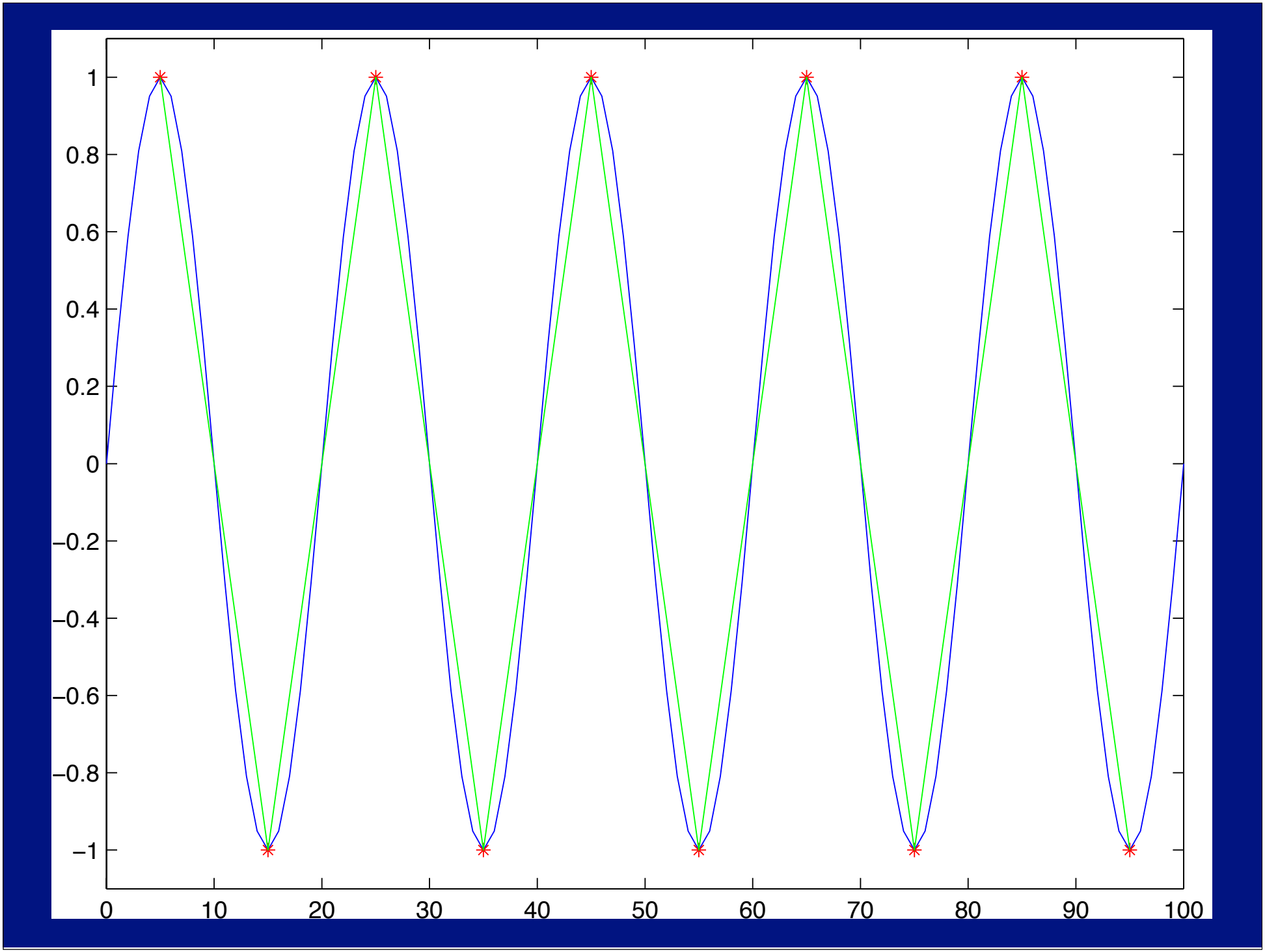
Fundamental facts

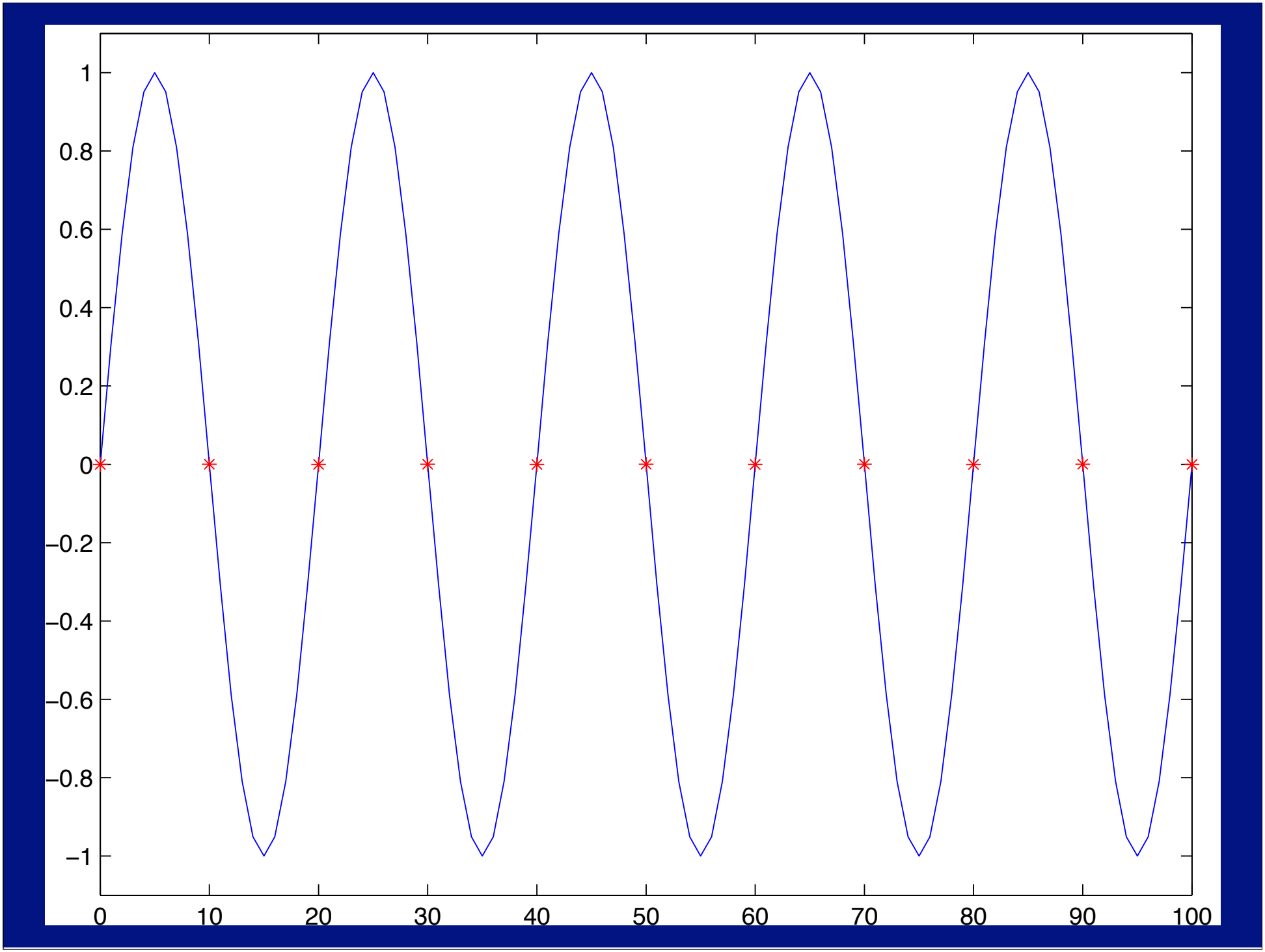
- A sine wave will alias if sampled less often than twice per period

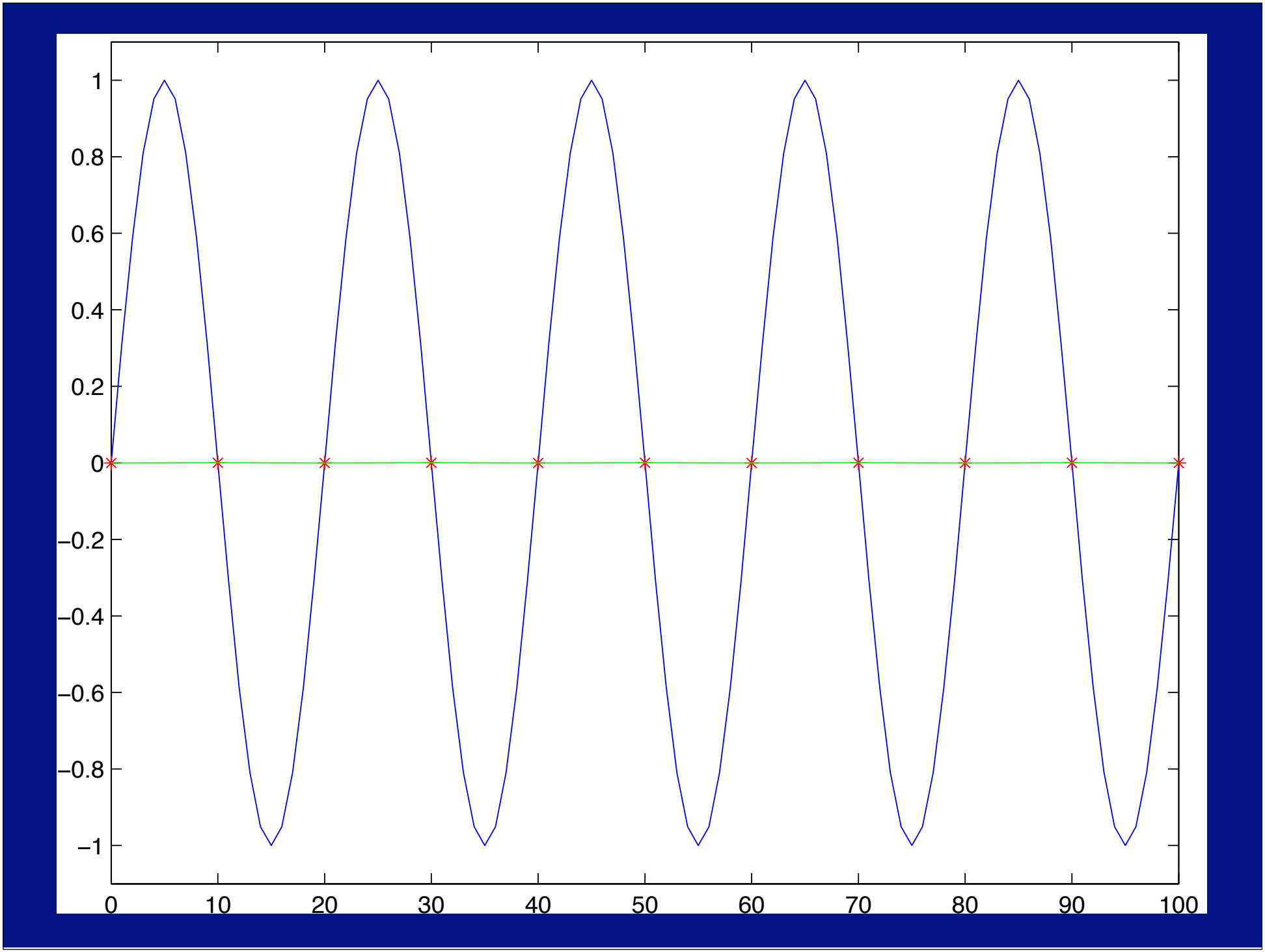

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

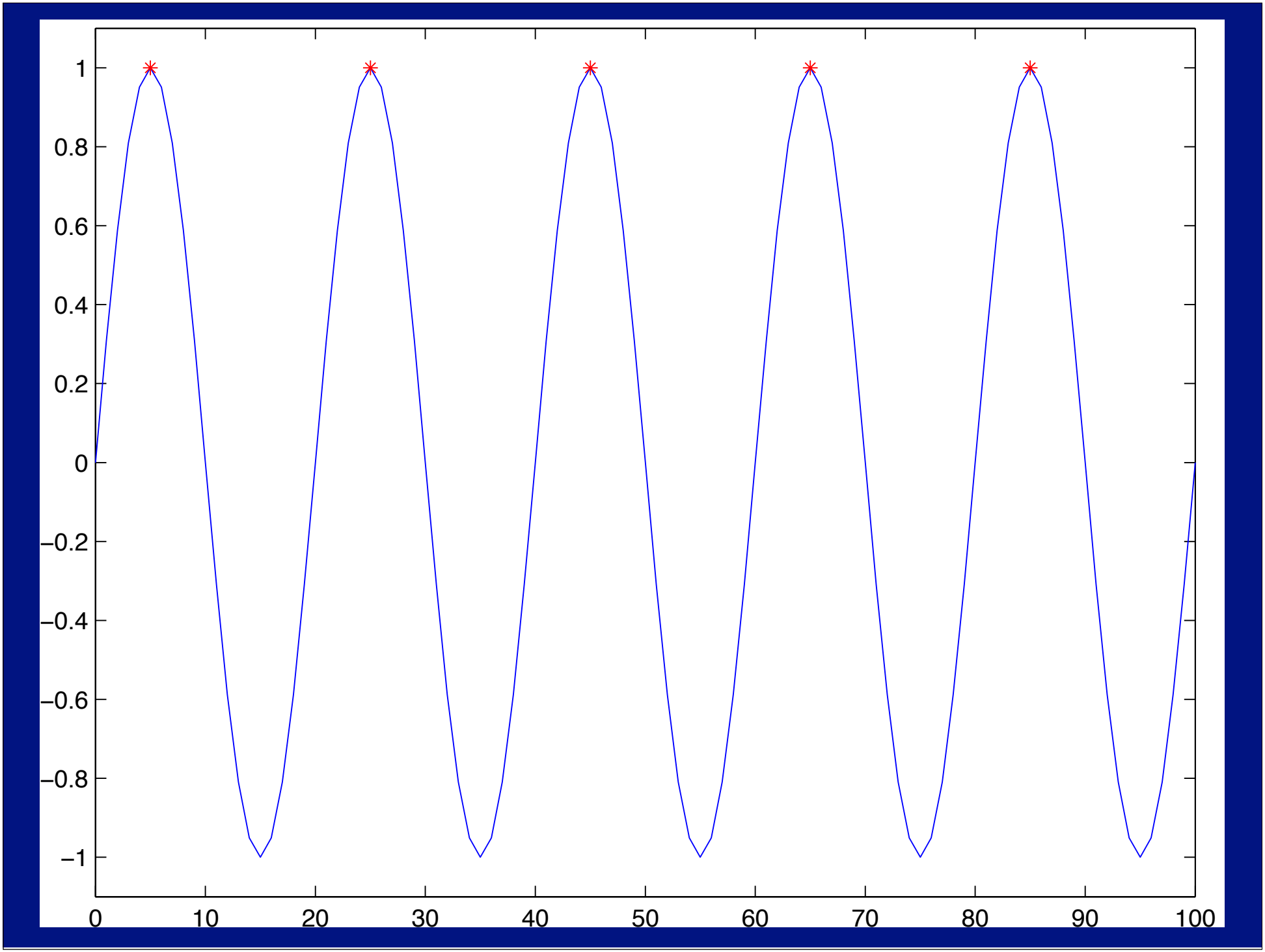

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

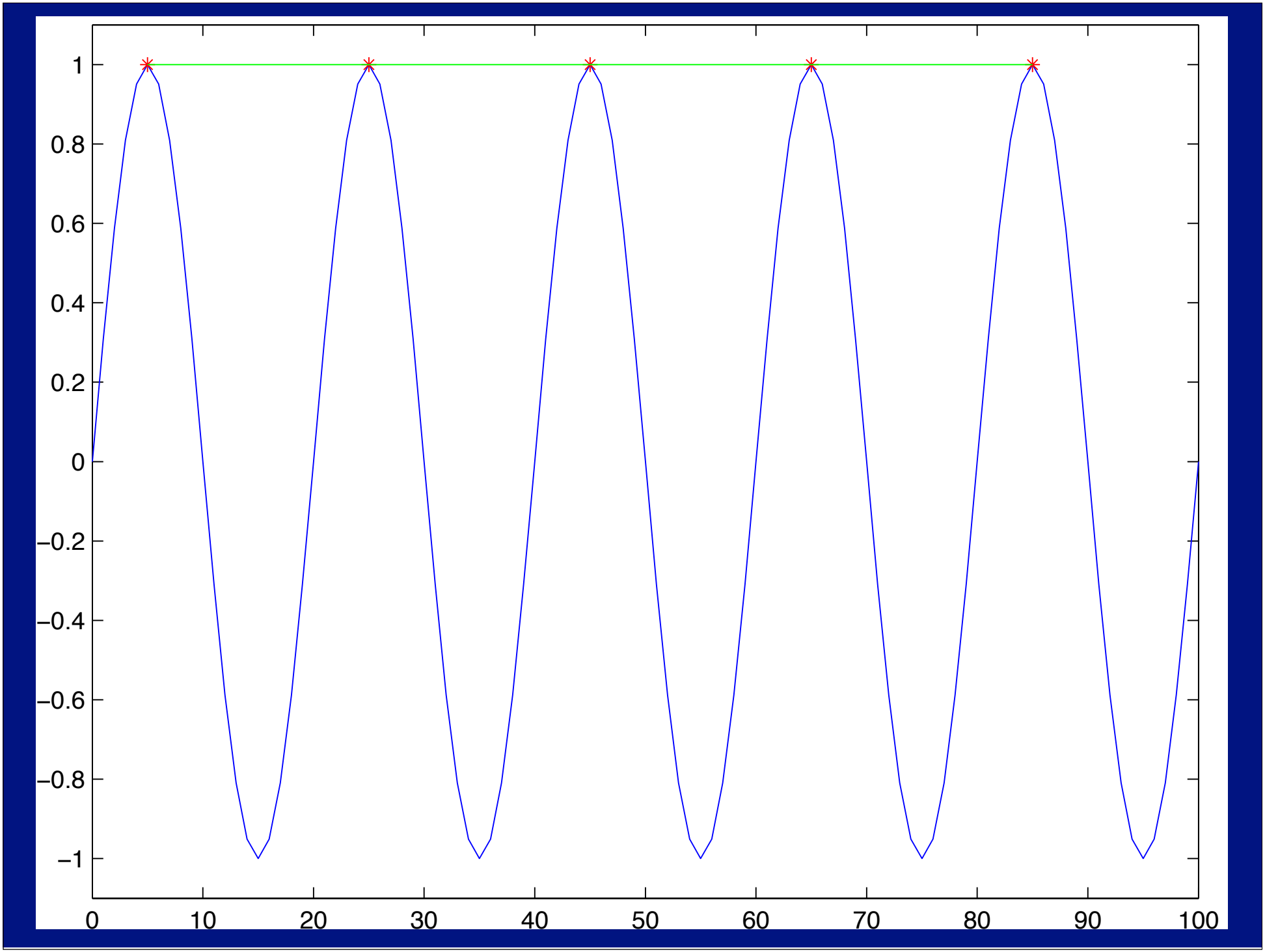

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

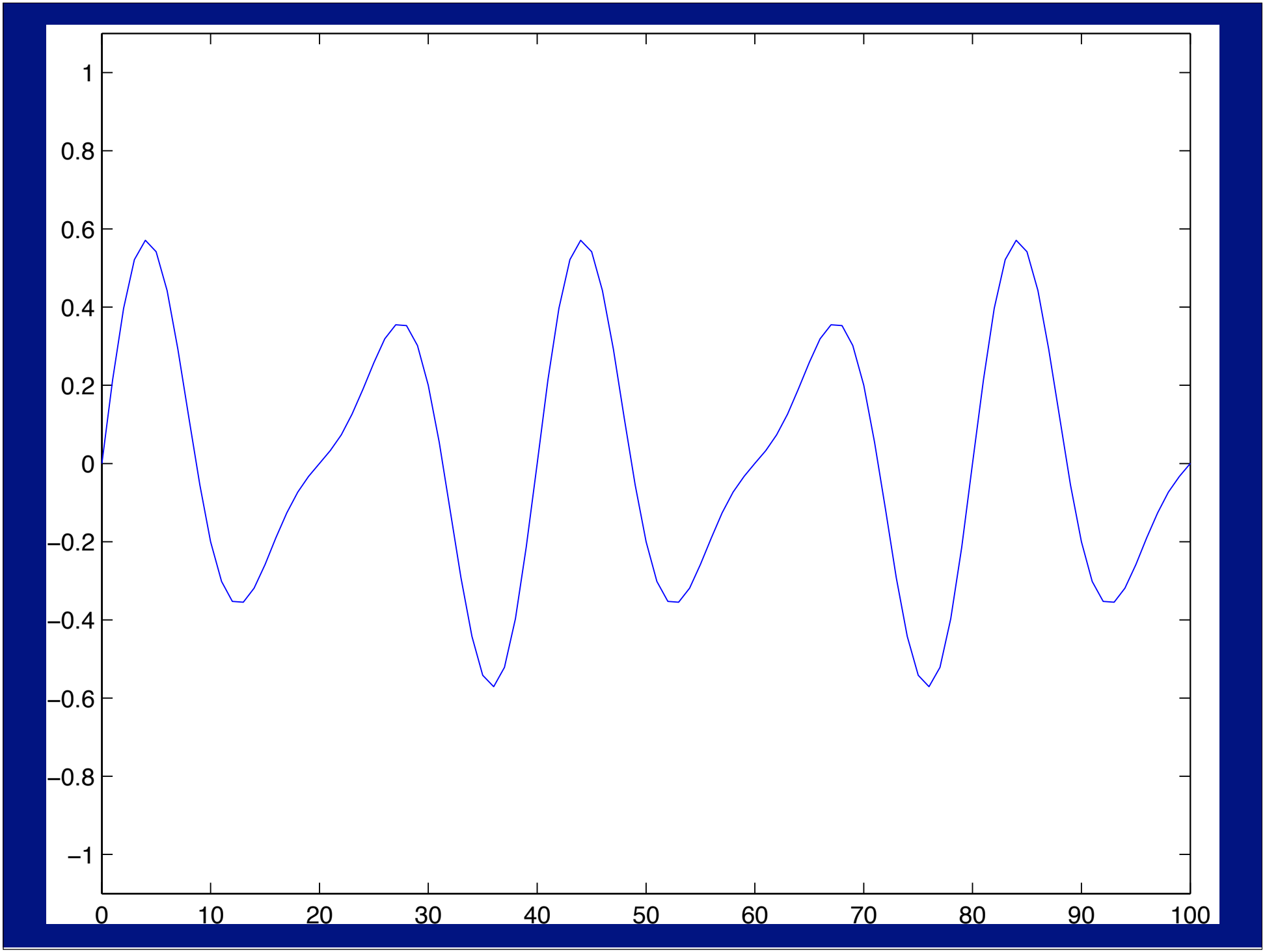

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

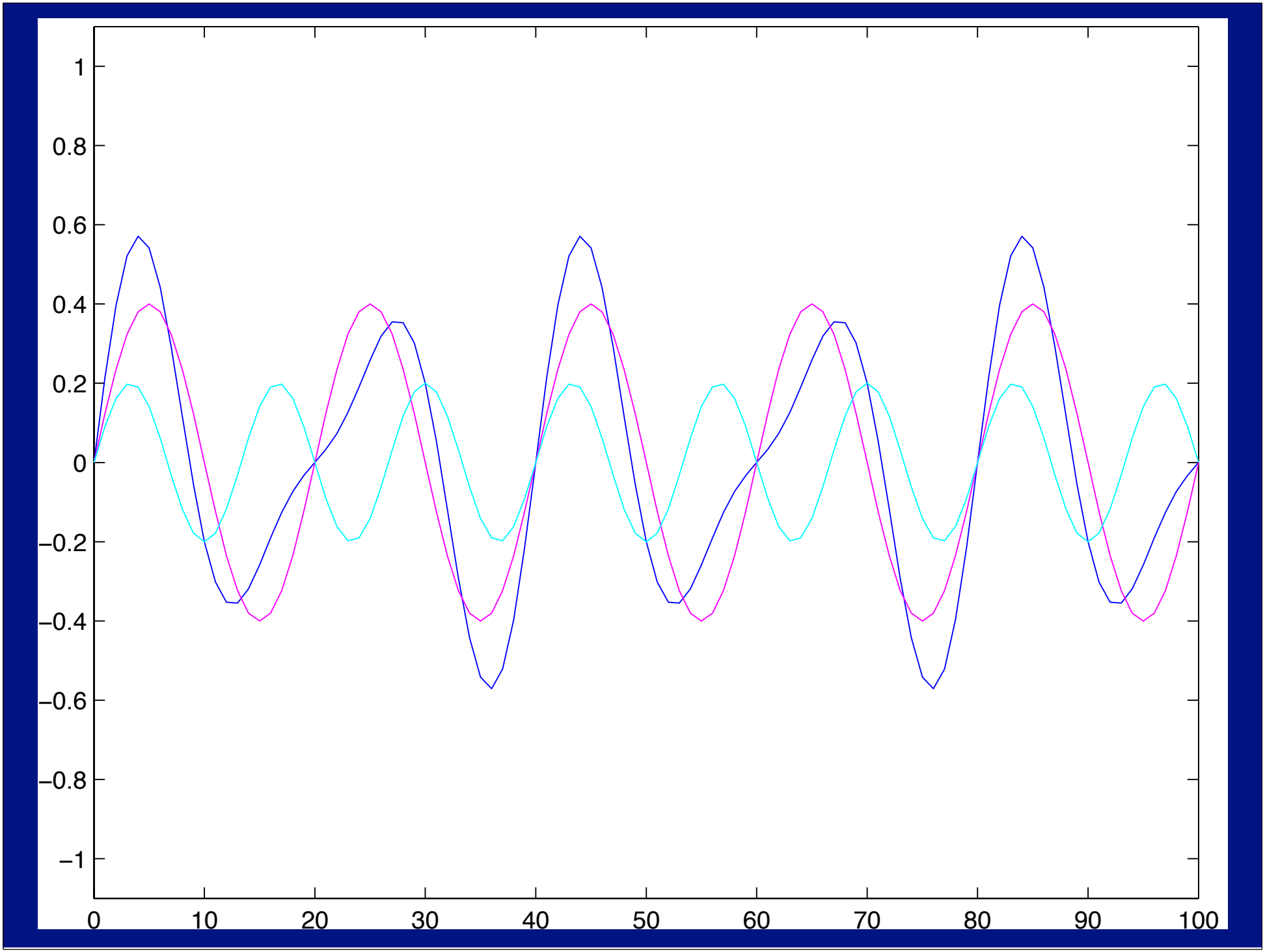

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

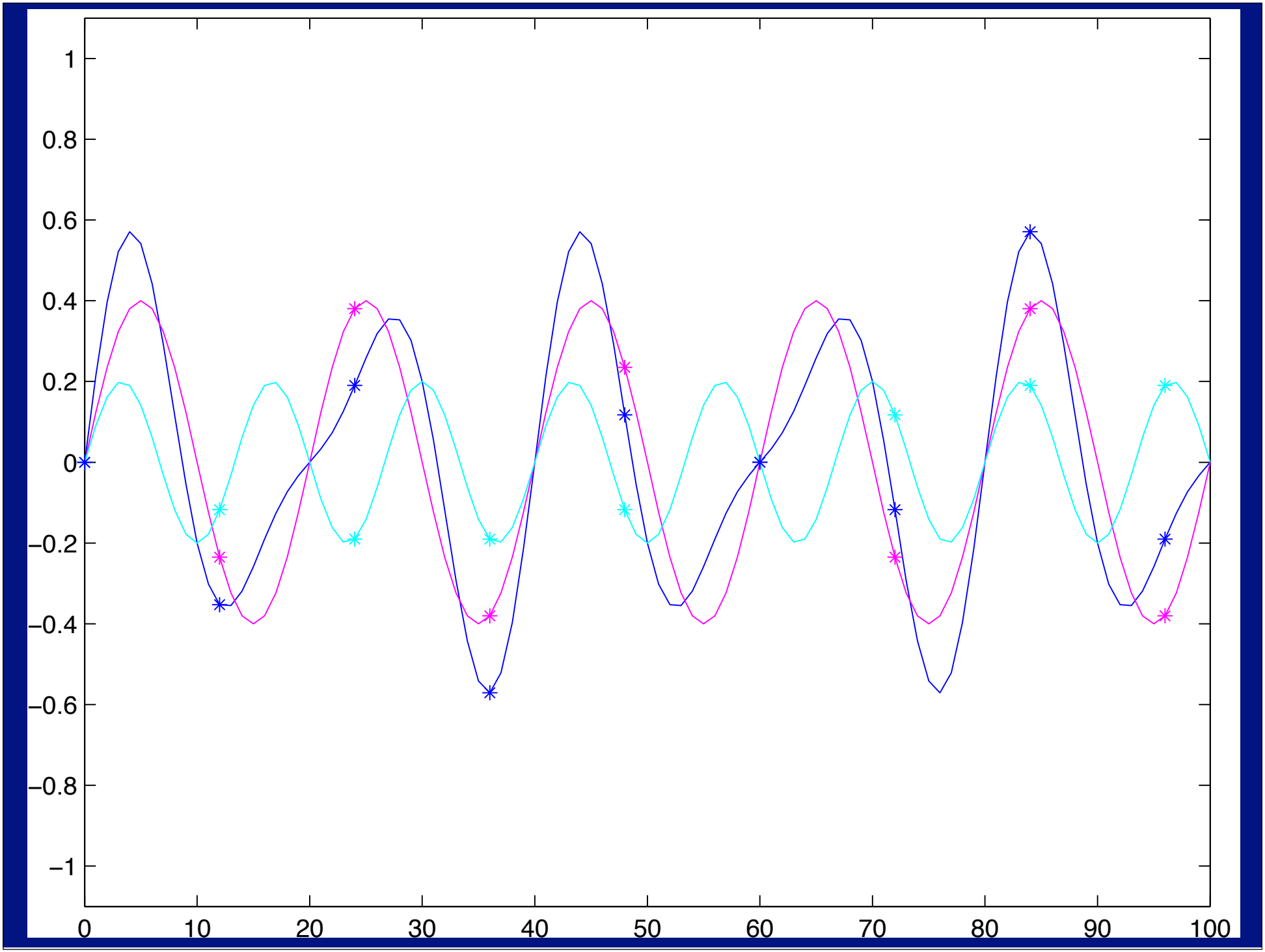

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

Fundamental facts

- Sample(A+B)=Sample(A)+Sample(B)
  - if a signal contains a high frequency sine wave, it will alias


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

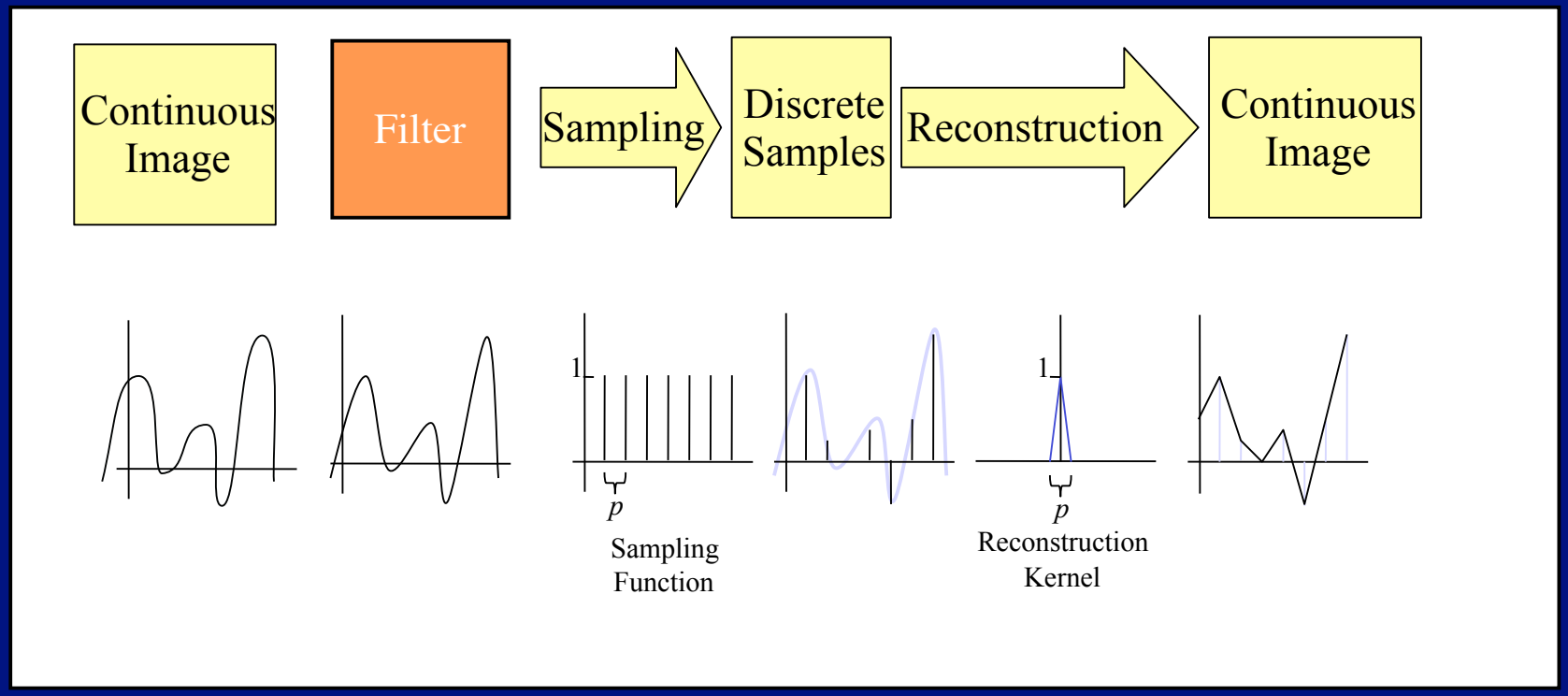

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
10
20
30
40
50
60
70
80
90
100

Weapons against aliasing

- Filtering
  - or smoothing
  - take the signal, reduce the fast-changing/high-frequency content
  - can do this by weighted local averaging

Prefiltering (Ideal case)



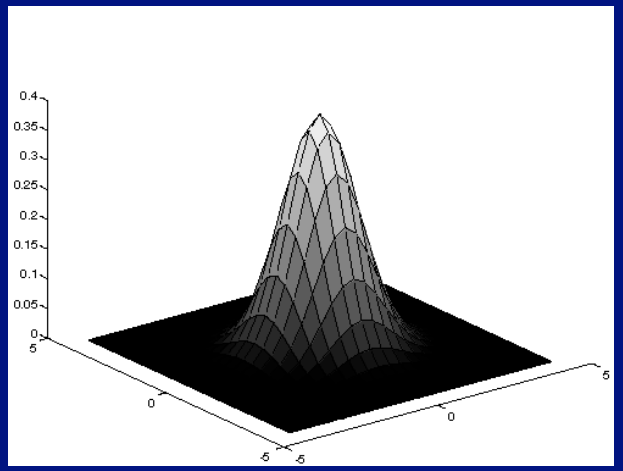
Smoothing by Averaging

$$N_{ij} = \frac{1}{N} \Sigma_{uv} O_{i+u,j+v}$$

where u, v, is a window of N pixels in total centered at 0, 0

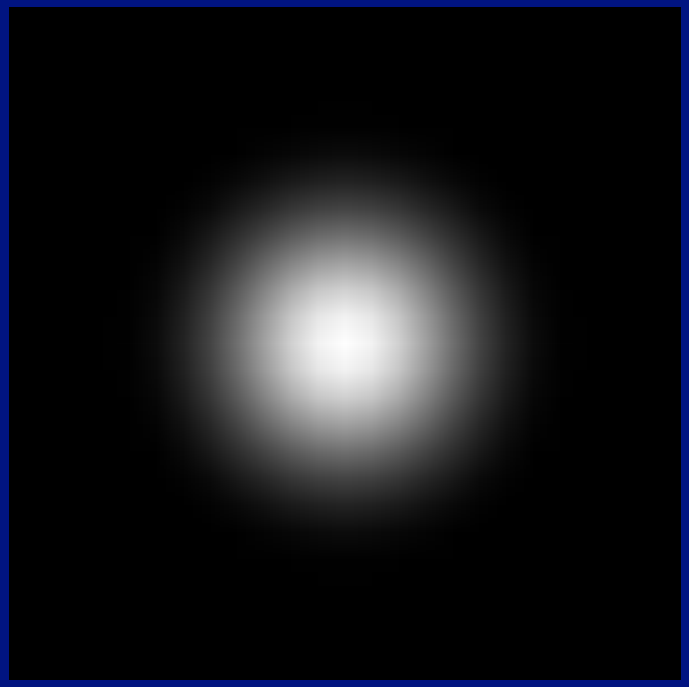
Smoothing with a Gaussian

- Notice “ringing”
  - apparently, a grid is superimposed
- Smoothing with an average actually doesn’t compare at all well with a defocussed lens
  - what does a point of light produce?



- A Gaussian gives a good model of a fuzzy blob

Gaussian filter kernel



$$K_{uv} = \left(\frac{1}{2\pi\sigma^2}\right) \exp\left(\frac{-\left[u^2+v^2\right]}{2\sigma^2}\right)$$

We're assuming the index can take negative values

Smoothing with a Gaussian

$$N_{ij} = \sum_{uv} O_{i-u,j-v} K_{uv}$$

Notice the curious looking form

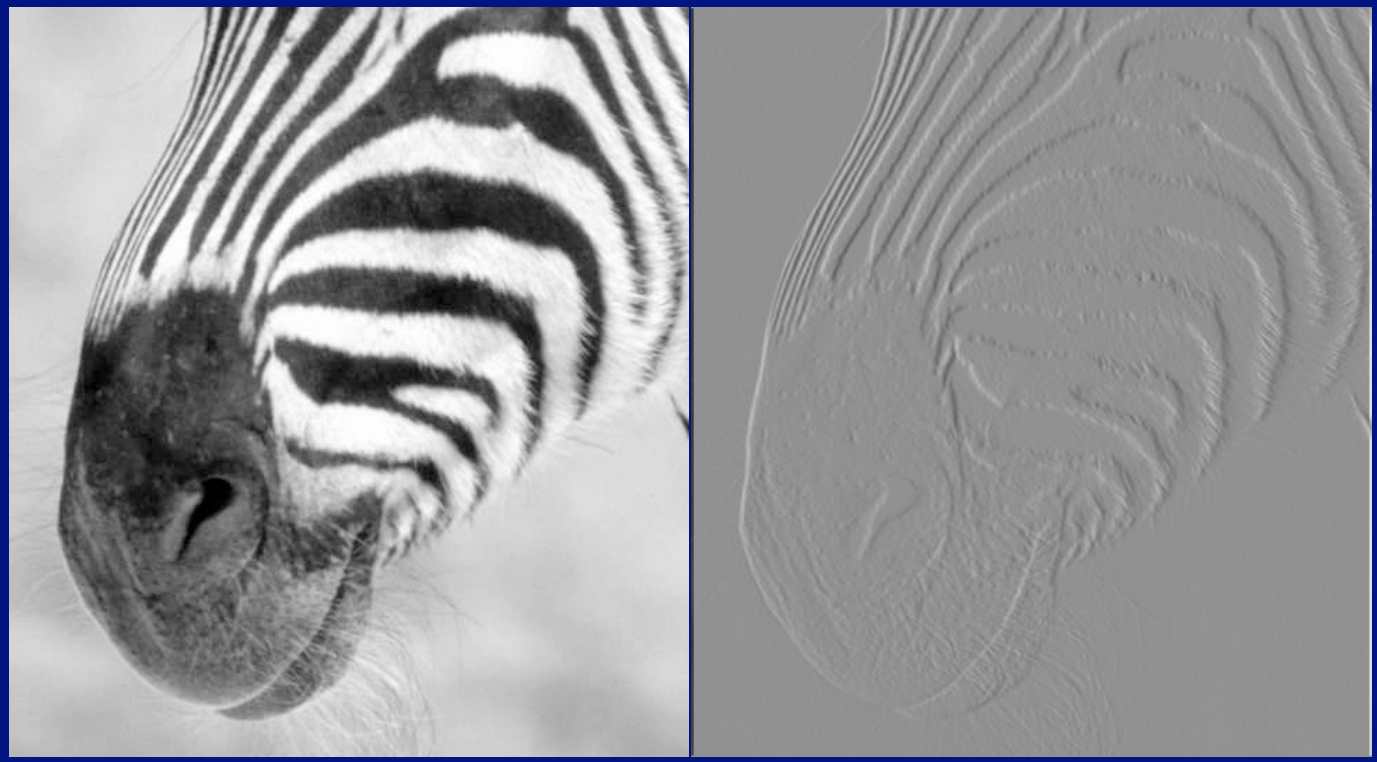
Matlab slide: convolution in 2D

```
%%
ker1=fspecial('gaussian', 9, 1);
figure(1); imshow(ker1);
figure(2); imshow(ker1/max(max(ker1)));
ker2=fspecial('gaussian', 21, 4);
figure(3); imshow(ker2);
figure(4); imshow(ker2/max(max(ker2)));
spi=double(reshape(sum(wind, 3), size(wind, 1), size(wind, 2)))/(3*256);
sm1sp=conv2(spi, ker1, 'same');
sm2sp=conv2(spi, ker2, 'same');
figure(5); imshow([spi, sm1sp, sm2sp]);
```

Linear Filters

- Example: smoothing by averaging
  - form the average of pixels in a neighbourhood
- Example: smoothing with a Gaussian
  - form a weighted average of pixels in a neighbourhood
- Example: finding a derivative
  - form a weighted average of pixels in a neighbourhood

Finding derivatives



$$N_{ij} = \frac{1}{\Delta x}(I_{i+1,j} - I_{ij})$$

Convolution

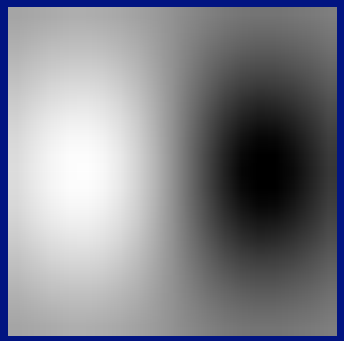
- Each of these involves a weighted sum of image pixels
- The set of weights is the same
  - we represent these weights as an image, H
  - H is usually called the kernel
- Operation is called convolution
  - it's associative
- Any linear shift-invariant operation can be represented by convolution
  - linear: G(k f)=k G(f)
  - shift invariant: G(Shift(f))=Shift(G(f))
  - Examples:
    - smoothing, differentiation, camera with a reasonable, defocussed lens system

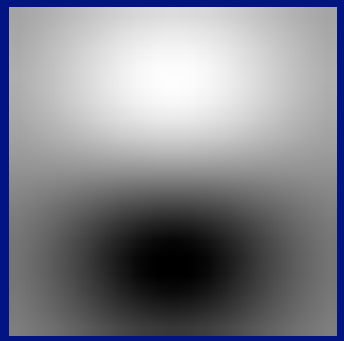
$$N_{ij} = \sum_{uv} H_{uv} O_{i-u,j-v}$$

Filters are templates

$$N_{ij} = \sum_{uv} H_{uv} O_{i-u,j-v}$$

- At one point
  - output of convolution is a (strange) dot-product
- Filtering the image involves a dot product at each point
- Insight
  - filters look like the effects they are intended to find
  - filters find effects they look like





Smoothing reduces noise

- Generally expect pixels to “be like” their neighbours
  - surfaces turn slowly
  - relatively few reflectance changes
- Expect noise to be independent from pixel to pixel
  - Implies that smoothing suppresses noise, for appropriate noise models
- Scale
  - the parameter in the symmetric Gaussian
  - as this parameter goes up, more pixels are involved in the average
  - and the image gets more blurred
  - and noise is more effectively suppressed

$$K_{uv} = \left(\frac{1}{2\pi\sigma^2}\right) \exp\left(\frac{-\left[u^2+v^2\right]}{2\sigma^2}\right)$$

Representing image changes: Edges

- Idea:
  - points where image value change very sharply are important
    - changes in surface reflectance
    - shadow boundaries
    - outlines
- Finding Edges:
  - Estimate gradient magnitude using appropriate smoothing
  - Mark points where gradient magnitude is
    - Locally biggest and
    - big

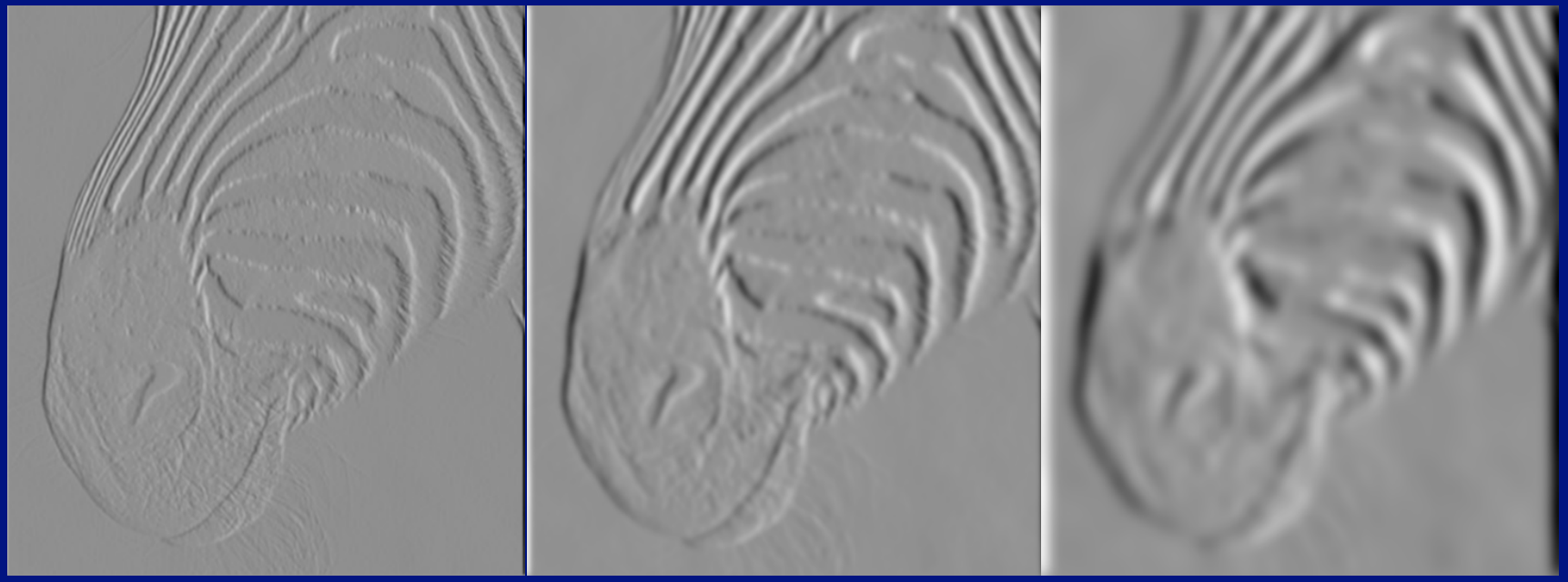
Matlab slide: gradients

```
zebra=imread('zebra.jpg');
zebrai=double(reshape(sum(zebra, 3), size(zebra, 1), size(zebra, 2)))/
(3*256);
zebras1i=conv2(zebrai, ker1, 'same');
zebras2i=conv2(zebrai, ker2, 'same');
[zix, ziy]=gradient(zebrai);
[zis1x, zis1y]=gradient(zebras1i);
[zis2x, zis2y]=gradient(zebras2i);
zigm=sqrt(zix.^2+ziy.^2);
zigs1m=sqrt(zis1x.^2+zis1y.^2);
zigs2m=sqrt(zis2x.^2+zis2y.^2);
figure(6); imshow(zigm/max(max(zigm)));
figure(7); imshow(zigs1m/max(max(zigs1m)));
figure(8); imshow(zigs2m/max(max(zigs2m)));
```

Matlab slide: smoothed gradients

```
zebra=imread('zebra.jpg');
zebrai=double(reshape(sum(zebra, 3), size(zebra, 1), size(zebra, 2)))/
(3*256);
zebras1i=conv2(zebrai, ker1, 'same');
zebras2i=conv2(zebrai, ker2, 'same');
[zix, ziy]=gradient(zebrai);
[zis1x, zis1y]=gradient(zebras1i);
[zis2x, zis2y]=gradient(zebras2i);
zigm=sqrt(zix.^2+ziy.^2);
zigs1m=sqrt(zis1x.^2+zis1y.^2);
zigs2m=sqrt(zis2x.^2+zis2y.^2);
figure(6); imshow(zigm/max(max(zigm)));
figure(7); imshow(zigs1m/max(max(zigs1m)));
figure(8); imshow(zigs2m/max(max(zigs2m)));
```

Scale affects derivatives

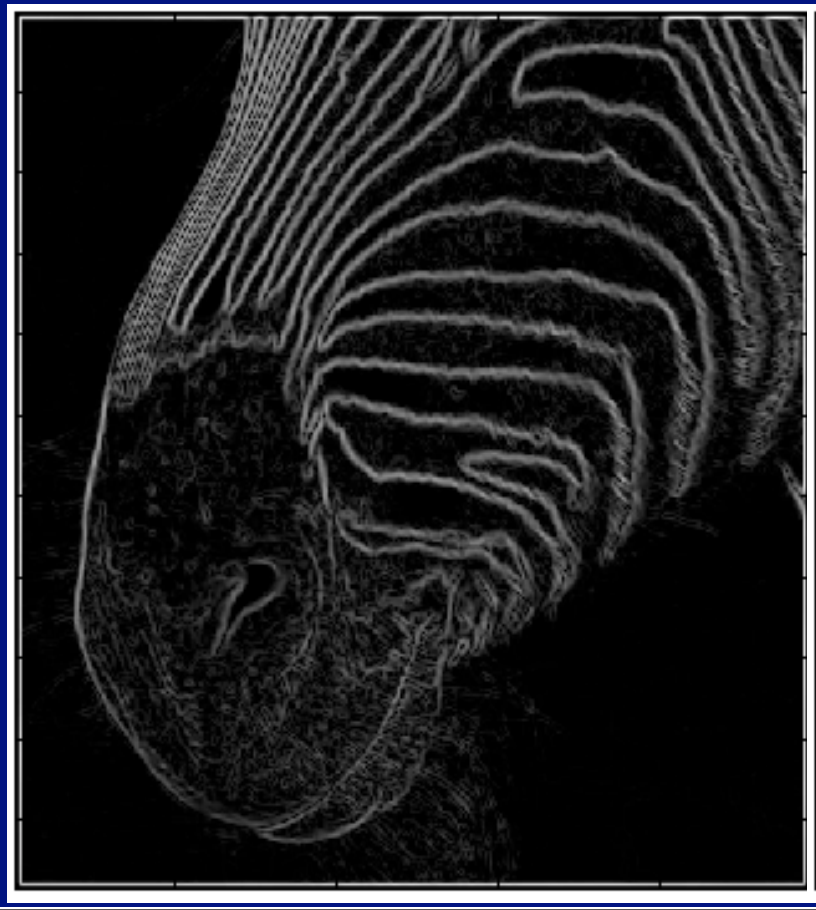


1 pixel

3 pixels

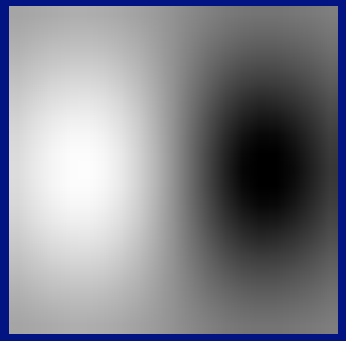
7 pixels

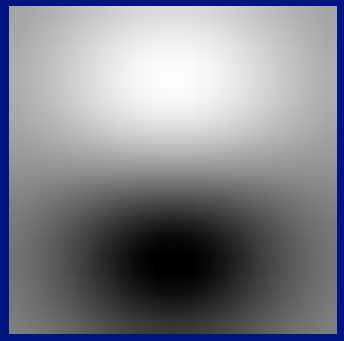
Scale affects gradient magnitude



Smoothing and Differentiation

- Issue: noise
  - smooth before differentiation
  - two convolutions to smooth, then differentiate?
  - actually, no - we can use a derivative of Gaussian filter





Matlab slide: orientations and arrow plots

```
step=7;
xw=[1:size(zix, 1)];
yw=[1:size(zix, 2)];
figure(9); clf;
imshow(zebrai);
hold on;
axis image;
quiver(yw(1:step:size(zix, 2)), ...
    xw(1:step:size(zix, 1)), ...
    zix(1:step:size(zix, 1), 1:step:size(zix, 2)),...
    ziy(1:step:size(zix, 1), 1:step:size(zix, 2)));
```

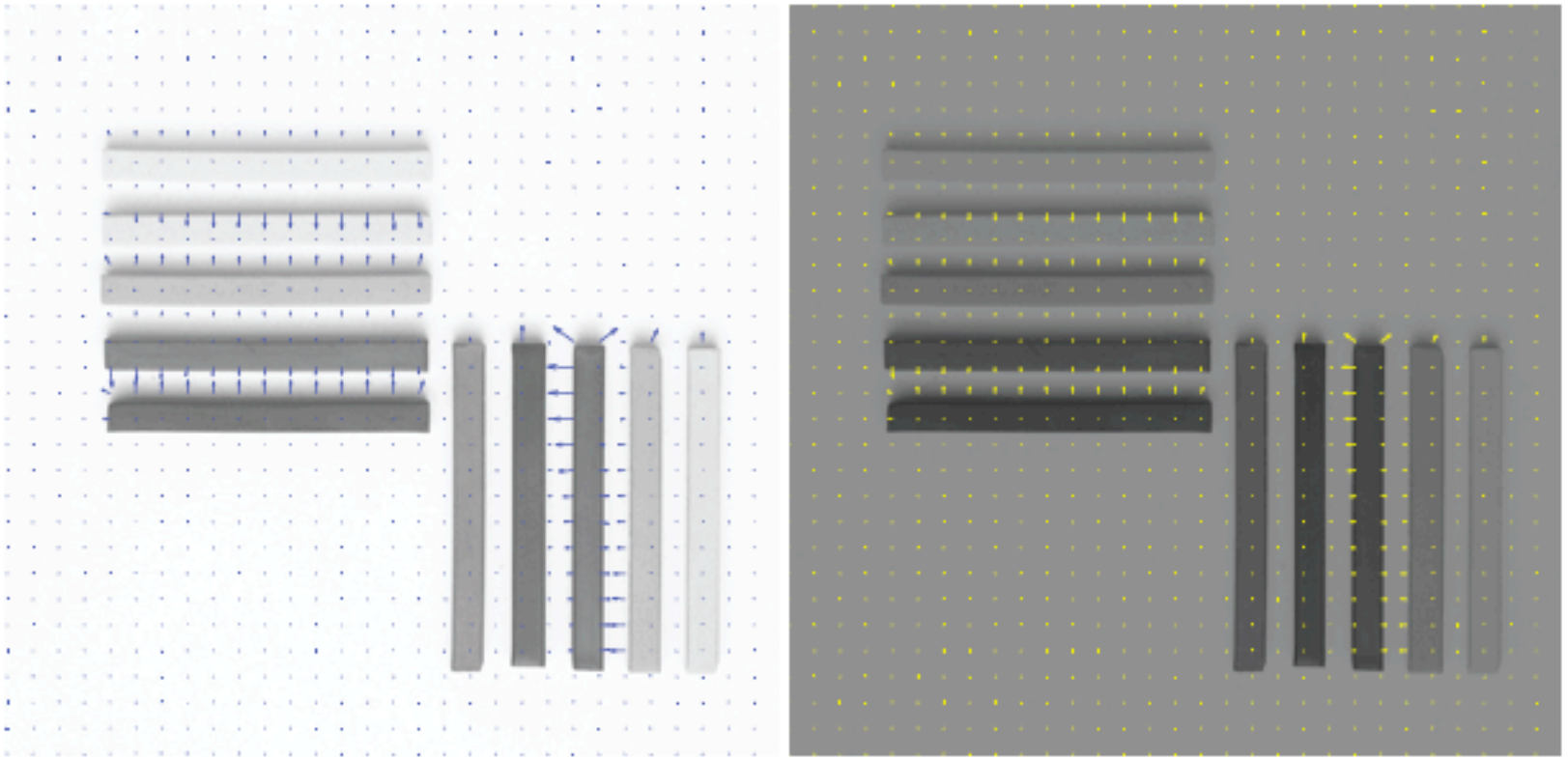


FIGURE 5.7: The magnitude of the image gradient changes when one increases or decreases the intensity. The orientation of the image gradient does not change; we have plotted every 10th orientation arrow, to make the figure easier to read. Note how the directions of the gradient arrows are fixed, whereas the size changes. *Philip Gatward © Dorling Kindersley, used with permission.*

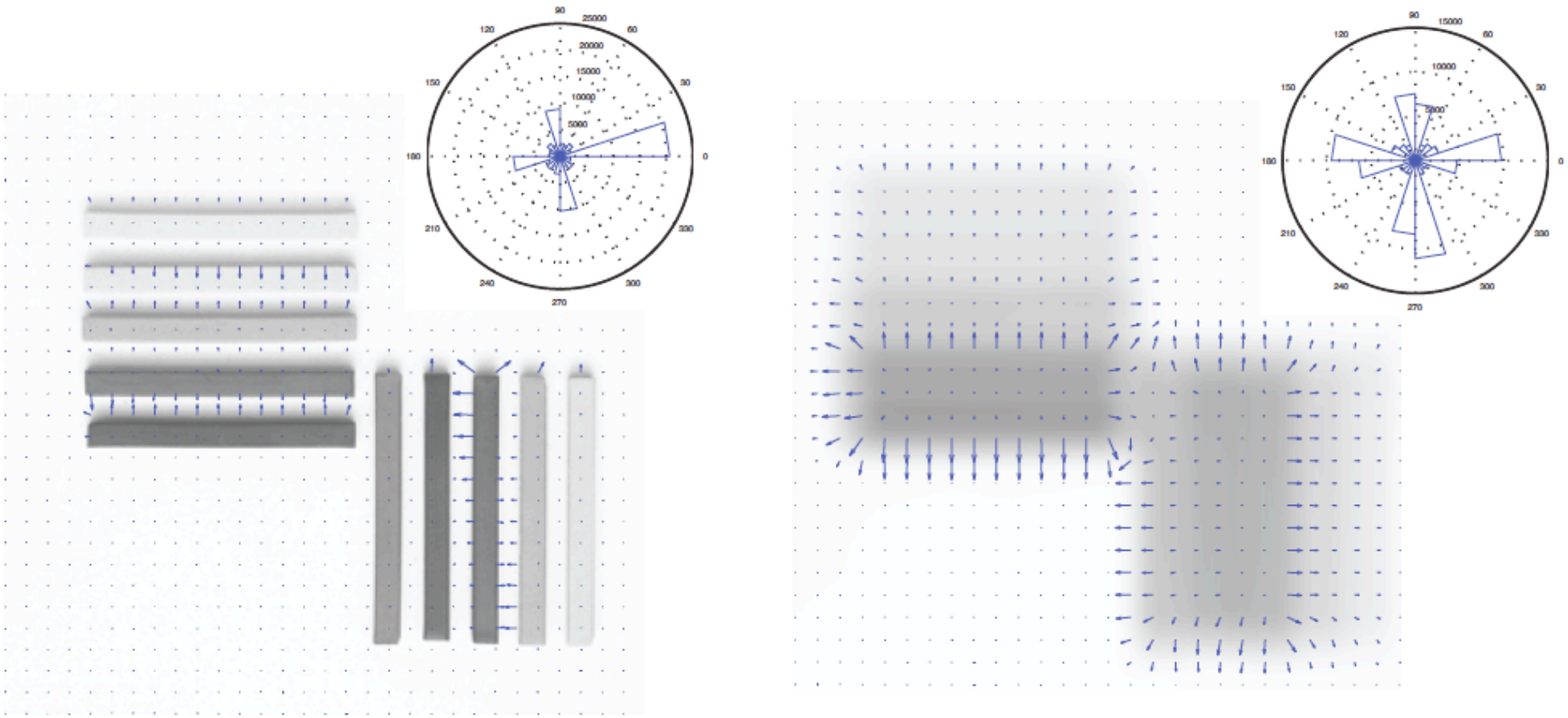


FIGURE 5.8: The scale at which one takes the gradient affects the orientation field. We show the overall trend of the orientation field by plotting a rose plot, where the size of a wedge represents the relative frequency of that range of orientations. **Left** shows an image of artists pastels at a fairly fine scale; here the edges are sharp, and so only a small set of orientations occurs. In the heavily smoothed version on the **right**, all edges are blurred and corners become smooth and blobby; as a result, more orientations appear in the rose plot. *Philip Gatward © Dorling Kindersley, used with permission.*

Matlab slide: rose plots

```
figure(10);
rose(reshape(atan2(ziy, zix), prod(size(zix)), 1));
```

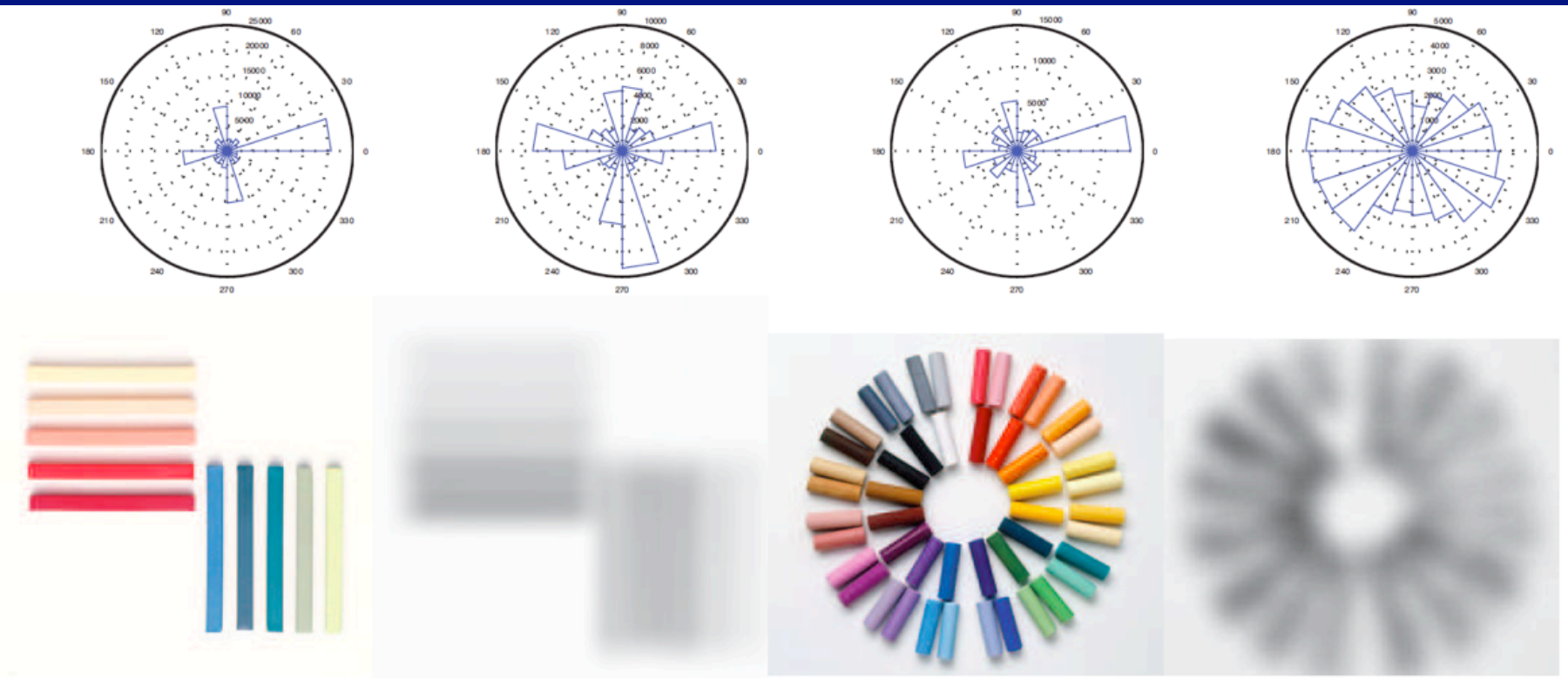


FIGURE 5.9: Different patterns have quite different orientation histograms. The **left** shows rose plots and images for a picture of artists pastels at two different scales; the **right** shows rose plots and images for a set of pastels arranged into a circular pattern. Notice how the pattern of orientations at a particular scale, and also the changes across scales, are quite different for these two very different patterns. *Philip Gatward © Dorling Kindersley, used with permission.*

Hog features

- Take a window
  - subdivide into boxes, each with multiple pixels
    - these might overlap
  - for each box, build a histogram of gradient orientations
    - possibly weighting by distance from center
    - possibly normalizing by intensity over the box
  - string these histograms together to a vector
- Extremely strong at spatial coding

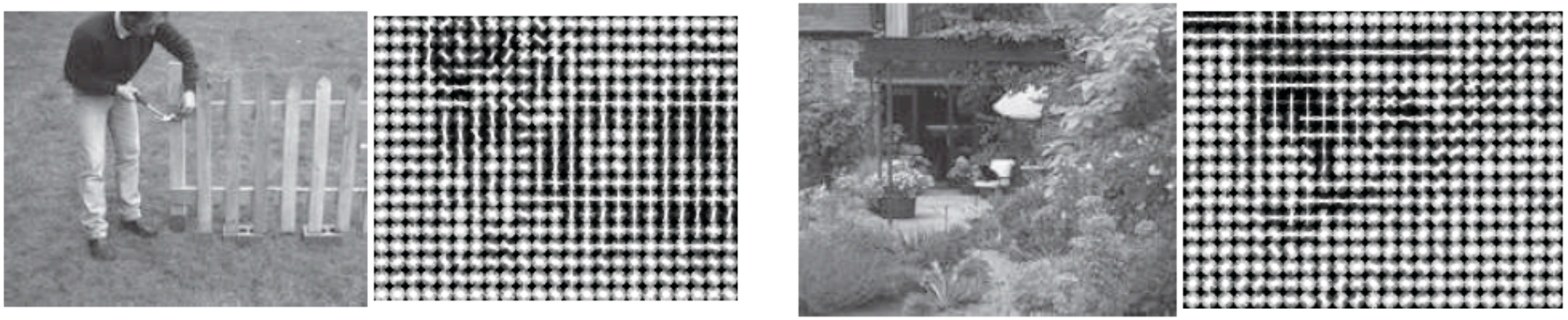


FIGURE 5.15: The HOG features for each the two images shown here have been visualized by a version of the rose diagram of Figures 5.7–5.9. Here each of the cells in which the histogram is taken is plotted with a little rose in it; the direction plotted is at right angles to the gradient, so you should visualize the overlaid line segments as edge directions. Notice that in the textured regions the edge directions are fairly uniformly distributed, but strong contours (the gardener, the fence on the **left**; the vertical edges of the french windows on the **right**) are very clear. This figure was plotted using the toolbox of Dollár and Rabaud. *Left:* © *Dorling Kindersley, used with permission. Right: Geoff Brightling* © *Dorling Kindersley, used with permission.*

Vlfeat pointer

We have sketched the most important feature constructions, but there is a huge range of variants. Performance is affected by quite detailed questions, such as the extent of smoothing when evaluating orientations. Space doesn't allow a detailed survey of these questions (though there's some material in Section 5.6), and the answers seem to change fairly frequently, too. This means we simply can't supply accurate recipes for building each of these features.

Fortunately, at time of writing, there are several software packages that provide good implementations of each of these feature types, and of other variations. Piotr Dollár and Vincent Rabaud publish a toolbox at `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`; we used this to generate several figures. VLFeat is a comprehensive open-source package that provides SIFT features, vector quantization by a variety of methods, and a variety of other representations. At time of writing, it could be obtained from `http://www.vlfeat.org/`. SIFT features are patented (Lowe 2004), but David Lowe (the inventor) provides a reference object code implementation at `http://www.cs.ubc.ca/~lowe/keypoints/`. Navneet Dalal, one of the authors of the original HOG feature paper, provides an implementation at `http://www.navneetdalal.com/software/`. One variant of SIFT is PCA-SIFT, where one uses principal components to reduce the dimension of the SIFT representation (Ke and Sukthankar 2004). Yan Ke, one of the authors of the original PCA-SIFT paper, provides an implementation at `http://www.cs.cmu.edu/~yke/pcasift/`. Color descriptor code, which computes visual words based on various color SIFT features, is published by van de Sande *et al.* at `http://koen.me/research/colordescriptors/`.

Image

HOG features

Positive terms
in linear classifier

Negative terms
in linear classifier