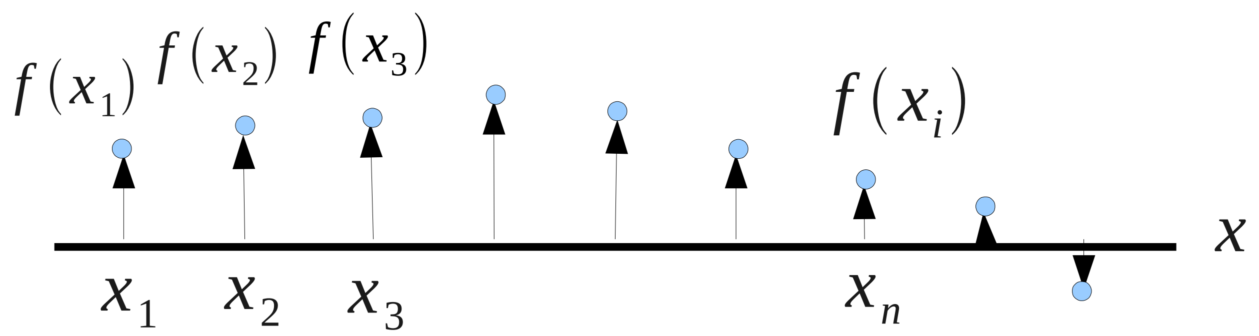
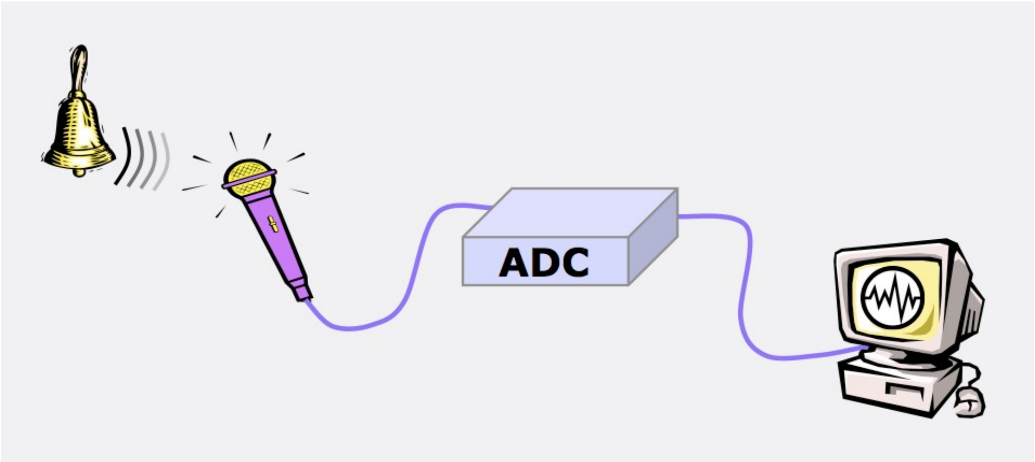
Sampled data

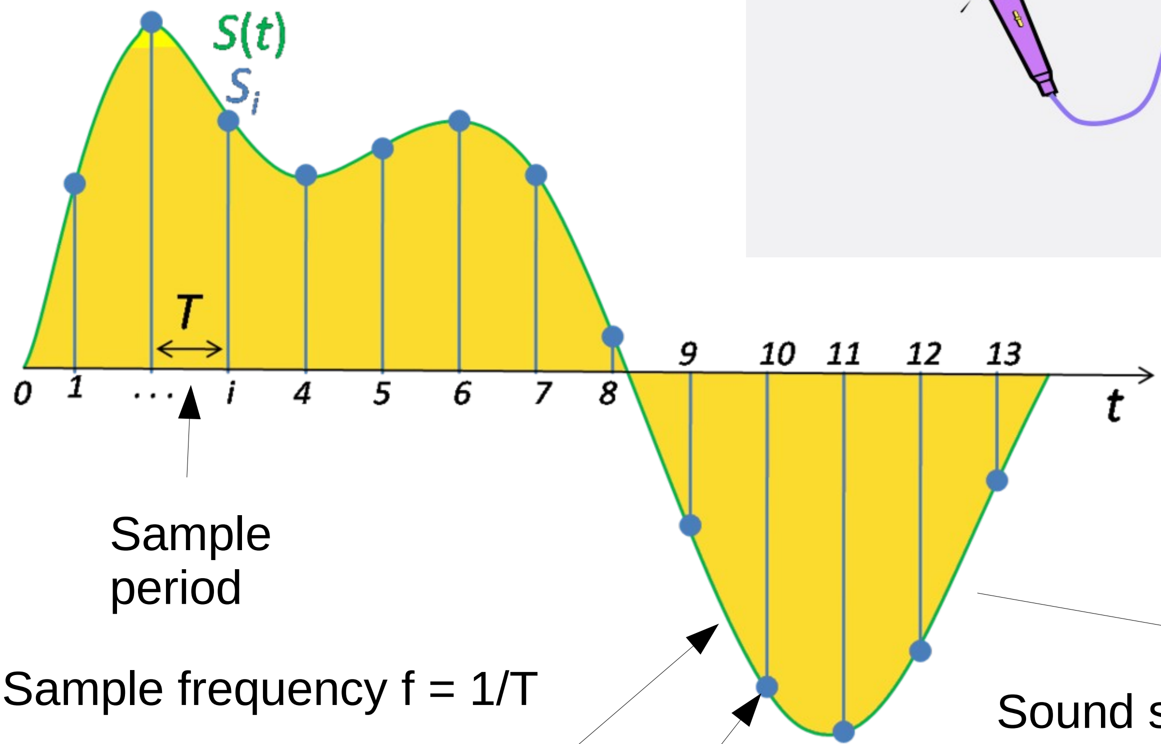


- Consider continuous function f(x).
- Sample the function at regular intervals.
- Sample points x_n
- The result is a vector of values of f: $[f_{1,} f_{2,} f_{3,} \cdots]$

Example: Digital audio

Sound wave captured by computer





Sample period

Sample frequency f = 1/T

Sound signal

Sound samples

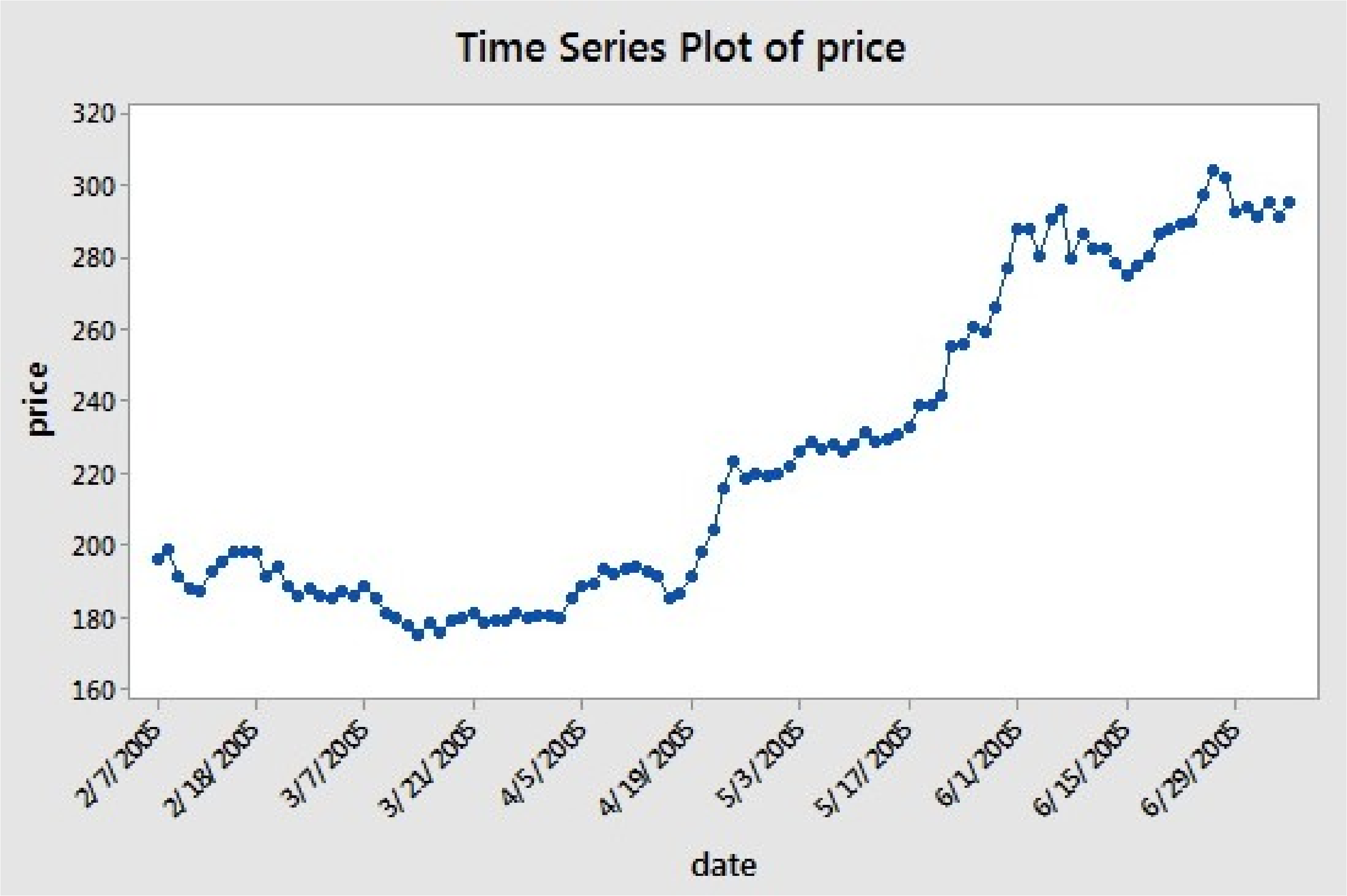
Sound samples represented as vector of numbers

0.3717
0.6901
0.9096
0.9989
0.9450
0.7557
0.4582
0.0951
-0.2817
-0.6182
-0.8660
-0.9898
-0.9718
-0.8146
-0.5406
-0.1893
0.1893

Callable function vs. Sampled data function

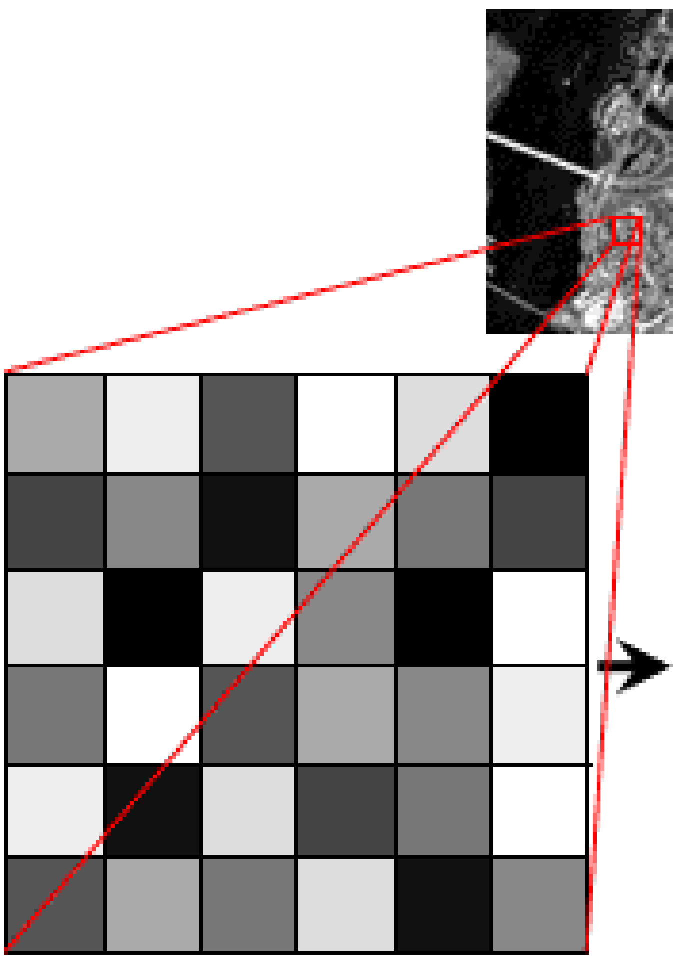
- We usually think of a function as callable.
- Now we need to think of a vector of data as another representation of a function.
- In general, sampling of real data is done using fixed (constant) period.
- For signal vector $f_n = f(t_n)$, here is always an implicit time vector tn lurking behind the scenes.
- Concept of streaming vs. Batch processing.

Example: stock prices vs. time



The idea is to treat the data vector as a function which varies in time.

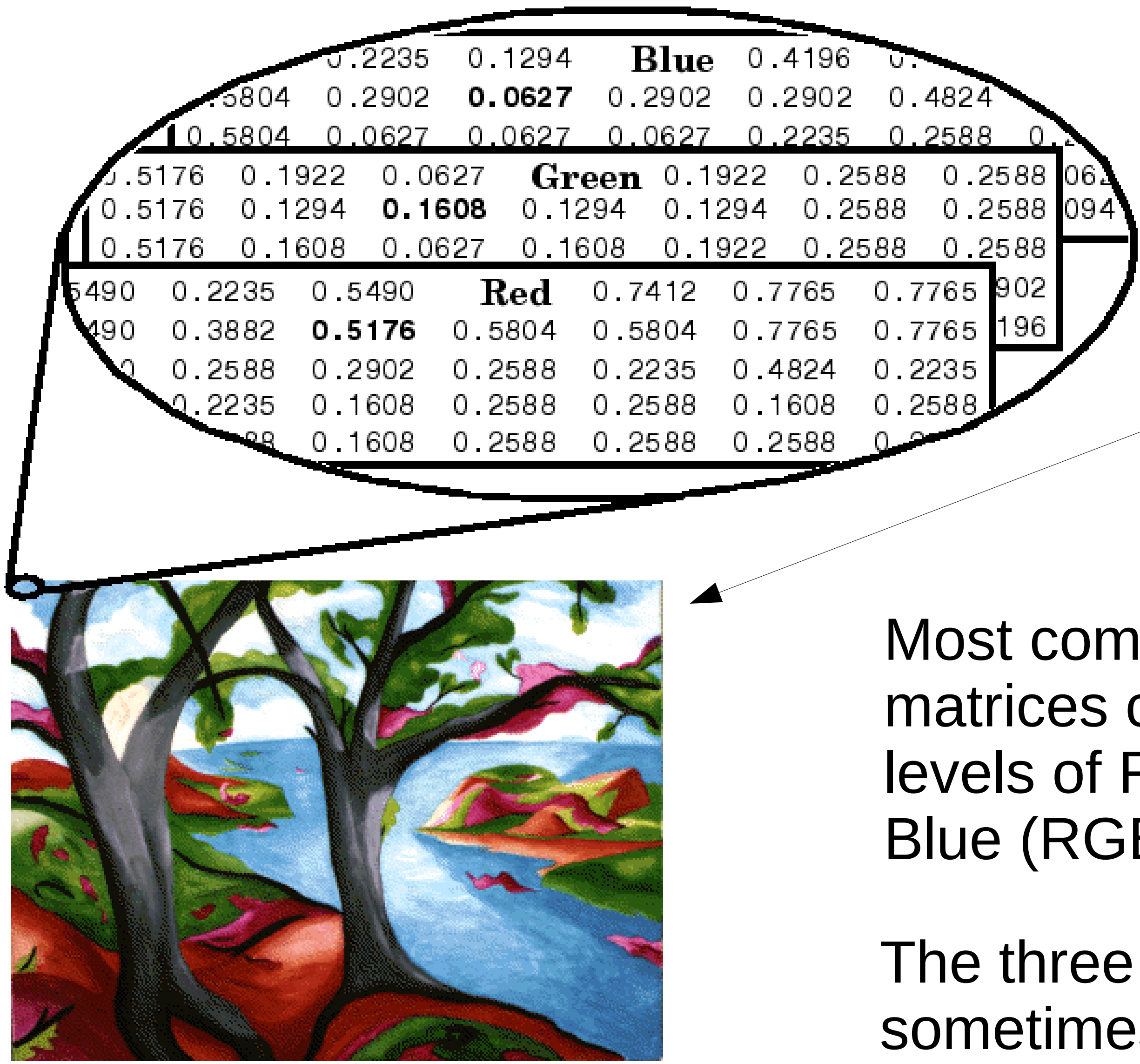
Example: Digital images (2D)



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

B&W image stored as matrix of “pixels” -- numbers signifying black/white level at each point.

Color images



Color image stored as three matrices of “pixels” -- numbers signifying intensity level at each point.

Most commonly, the three matrices correspond to levels of Red, Green, and Blue (RGB).

The three matrices are sometimes called “color planes”

Numerical derivatives and sampled data

- Use Taylor's series to derive:
  - Forward difference
  - Backward difference
  - Two-sided difference (symmetric difference)
- Note truncation error from each

Computing the first derivative

- Derive on blackboard:
  - Forward difference

Drop

$$\left.\frac{df}{dx}\right|_x = \frac{f(x+h)-f(x)}{h} - \frac{h}{2} f''(x) - \frac{h^2}{6} f'''(x) + \cdots$$

  - Backward difference

Drop

$$\left.\frac{df}{dx}\right|_x = \frac{f(x)-f(x-h)}{h} - \frac{h}{2} f''(x) + \frac{h^2}{6} f'''(x) + \cdots$$

  - Two-sided difference

Drop

$$\left.\frac{df}{dx}\right|_x = \frac{f(x+h)-f(x-h)}{2\mathrm{h}} - \frac{h^2}{6} f'''(x) + \cdots$$

Approximations using more points

Table 1. Compact central differencing formulas for the first derivative, f_0^{I}, with the leading term of its systematic error, for $j = 3(2)17$, where the number of data points j listed in the first column includes f_0. The term *compact* indicates use of the smallest possible number j of equidistant data. The results shown in tables 1 through 4 were computed with the spreadsheet approach illustrated in section 9.2.5 of ref. 6. For $j > 9$ this required higher-precision matrix inversion to get sufficiently accurate answers, for which we used Volpi's BigMatrix freeware, see ref. 6 section 11.9.

j	Formula for f_0^{I}	Leading term of systematic error
3	$(-f_{-1} + f_1)/(2\delta)$	$-f^{\mathrm{III}}\,\delta^2/6$
5	$(f_{-2} - 8f_{-1} + 8f_1 - f_2)/(12\delta)$	$+f^{\mathrm{V}}\,\delta^4/30$
7	$(-f_{-3} + 9f_{-2} - 45f_{-1} + 45f_1 - 9f_2 + f_3)/(60\delta)$	$-f^{\mathrm{VII}}\,\delta^6/140$
9	$(3f_{-4} - 32f_{-3} + 168f_{-2} - 672f_{-1} + 672f_1 - 168f_2 + 32f_3 - 3f_4)/(840\delta)$	$+f^{\mathrm{IX}}\,\delta^8/630$
11	$(-2f_{-5} + 25f_{-4} - 150f_{-3} + 600f_{-2} - 2100f_{-1} + 2100f_1 - 600f_2 + 150f_3 - 25f_4 + 2f_5)/(2520\delta)$	$+f^{\mathrm{XI}}\,\delta^{10}/2772$
13	$(5f_{-6} - 72f_{-5} + 495f_{-4} + 2200f_{-3}+7425f_{-2} - 23760f_{-1} + 23760f_1 - 7425f_2 + 2200f_3 - 495f_4 + 72f_5 - 5f_6)/(27720\delta)$	$+f^{\mathrm{XIII}}\,\delta^{12}/12012$
15	$(-15f_{-7} + 245f_{-6} - 1911f_{-5} + 9555f_{-4} - 35035f_{-3} + 105105f_{-2} - 315315f_{-1} + 315315f_1 - 105105f_2 + 35035f_3 - 9555f_4 + 1911f_5 - 245f_6 + 15f_7)/(360360\delta)$	$+f^{\mathrm{XV}}\,\delta^{14}/51480$
17	$(7f_{-8} - 128f_{-7} + 1120f_{-6} - 6272f_{-5} + 25480f_{-4} - 81536f_{-3} + 224224f_{-2} - 640640f_{-1} + 640640f_1 - 224224f_2 + 81536f_3 - 25480f_4 + 6272f_5 - 1120f_6 + 128f_7 - 7f_8)/(720720\delta)$	$+f^{\mathrm{XVII}}\,\delta^{16}/218790$

An improved numerical approximation for the first derivative[†]

ROBERT DE LEVIE

Chemistry Department, Bowdoin College, Brunswick ME 04011, USA

J. Chem. Sci., Vol. 121, No. 5, September 2009, pp. 935–950.

Second derivative

- Derived on blackboard

Drop

$$\left.\frac{d^2 f}{dx^2}\right|_x = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \cancel{\frac{h^2}{12} f^{(4)}(x) + \cdots}$$

- Truncation error is of order h^2

What h to use?

$$h=\sqrt{\epsilon(f(x))}$$

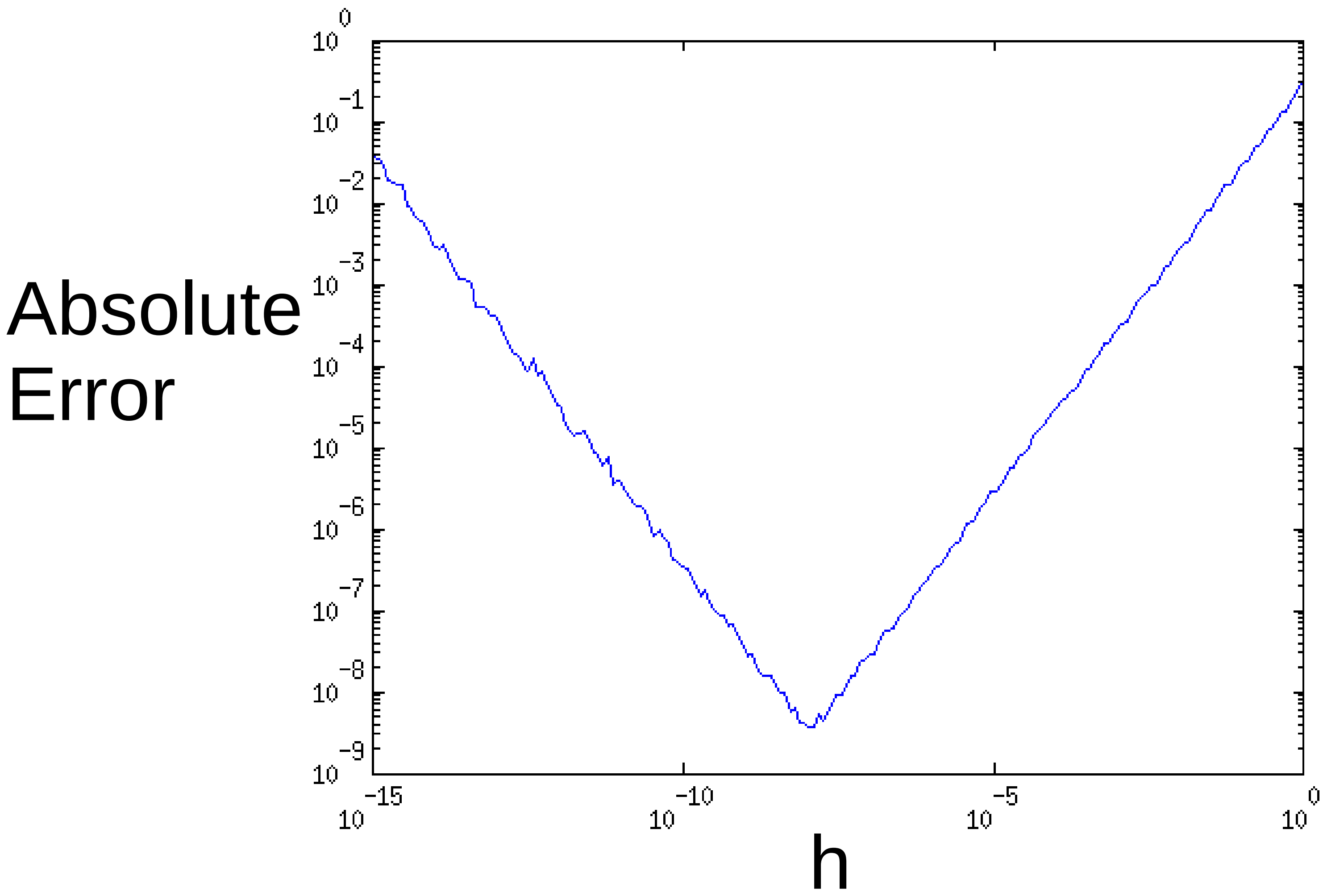
- Use Taylor's series
- Show truncation and round-off error
- Minimize total error to find optimal h.
- Derivation on blackboard
- Recommended values (one sided derivative):
  - If using doubles, and f(x) is near 1, use h = 1e-8
  - If using singles and f(x) is near 1, use h = 3e-4.

Demonstration

- Loop over h values
- Compute finite-difference derivative of yc = sin(x) for a bunch of random x values using this h value.
- Compute analytic derivative yt = cos(x).
- Compute average error mean(yt-yc) over different random x values.
- At end of loop, plot error vs. h.

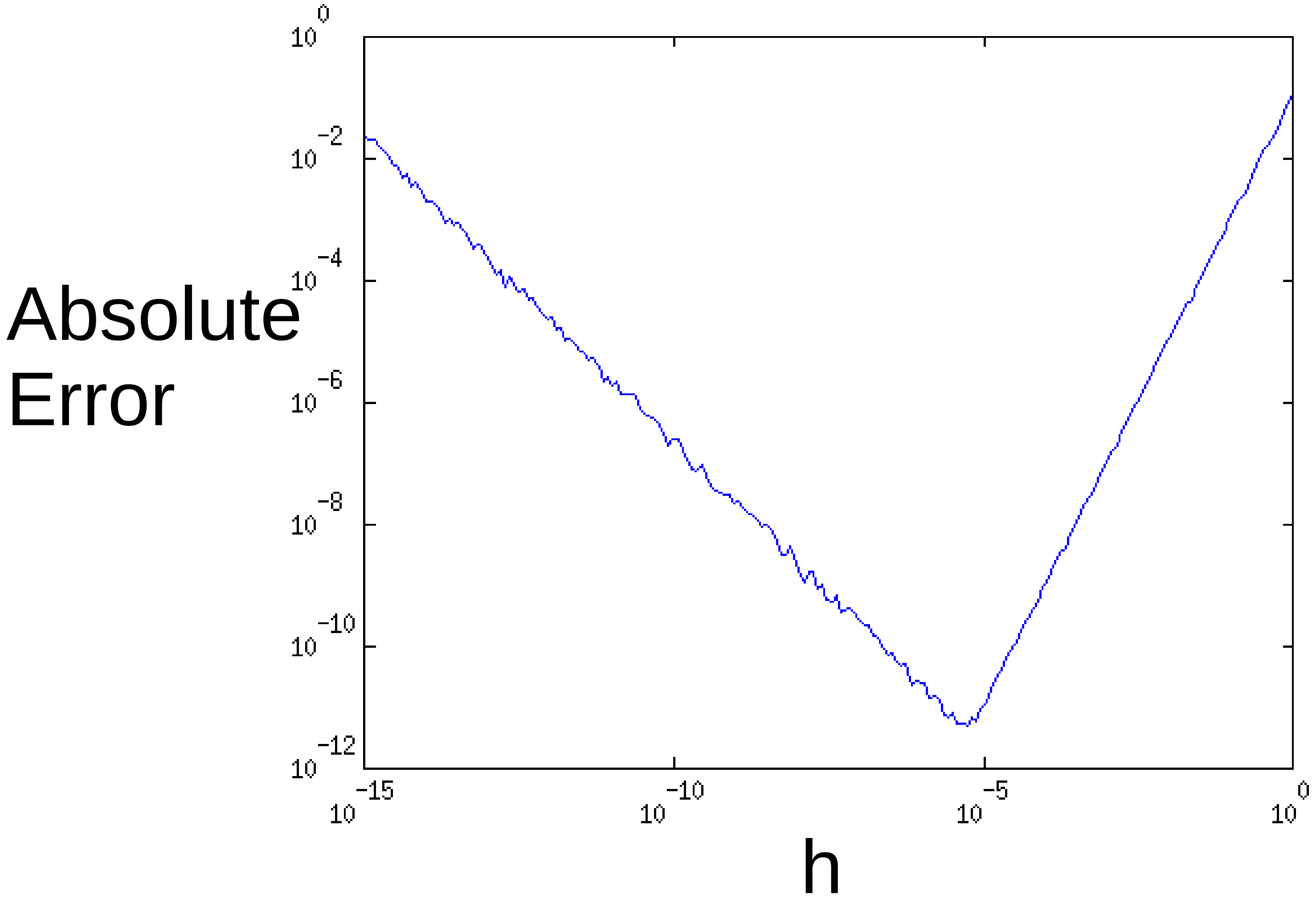
```
function [h_vec, err_vec] = test_derivative()
  start = -15;   % Start at 1e-15
  stop = 0;      % Stop at 1e0

  % Vector of h values to test
  h_vec = logspace(start, stop, 200);
  h_length = length(h_vec);

  % Pre-initialize error vector which will get filled in below.
  err_vec = zeros(1, h_length);

  % This is number of times to generate random x and compute
  % numerical derivative at that point to create average error.
  Nsamples = 50;

  % Main loop -- compute average error for each value of h in h_vec.
  for h_idx = 1:h_length
    h = h_vec(h_idx);

    % Create a row vector of random x values
    x = 1*randn(1, Nsamples);
    computed = derivative(@sin, x, h);
    true = cos(x);
    err = mean(computed-true);

    err_vec(h_idx) = err/Nsamples;
  end

  % plot results.
  loglog(h_vec, abs(err_vec))

end
```

One sided derivative

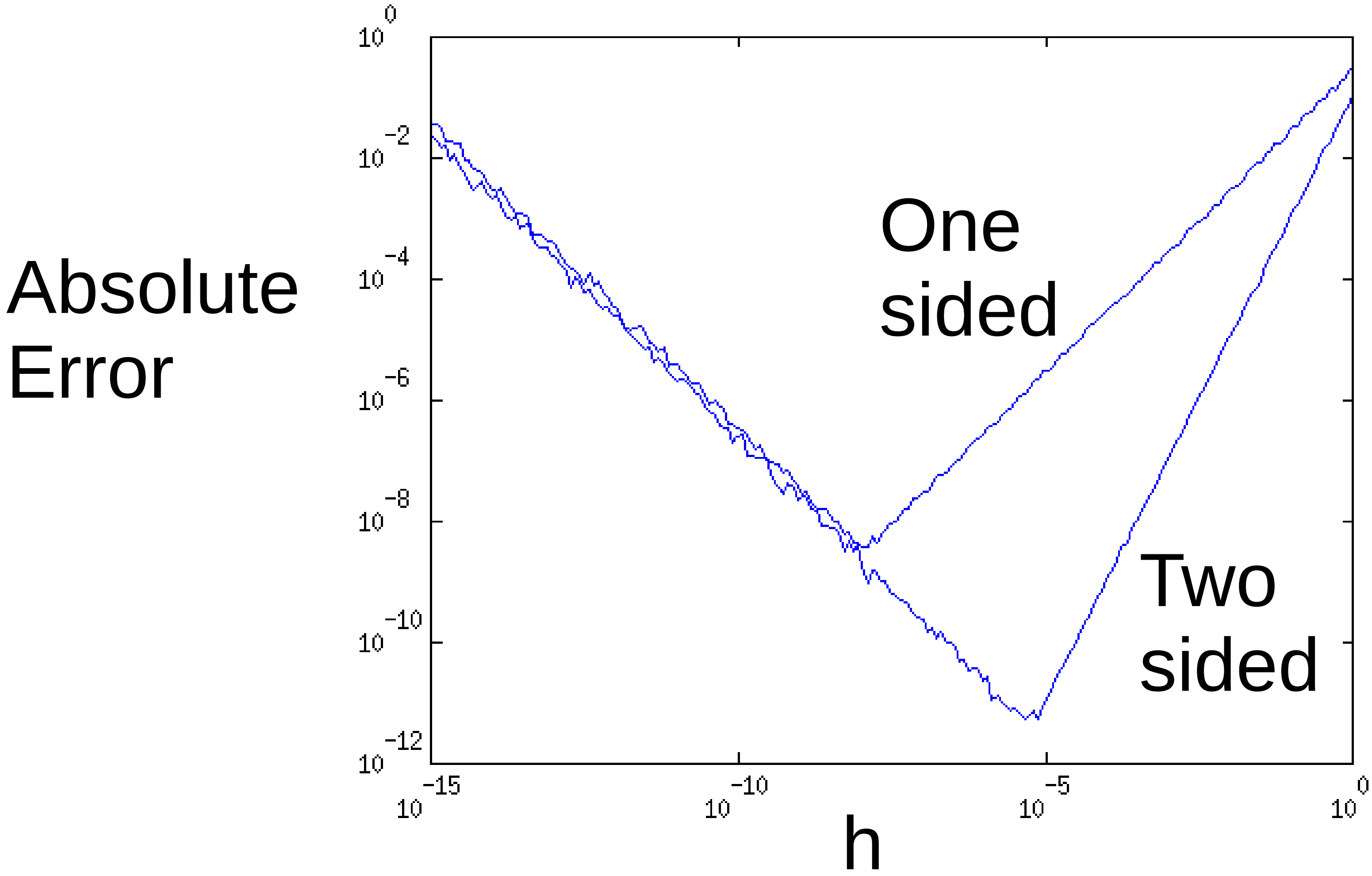


If using doubles, and f(x) is near 1, use h = 1e-8

Two sided derivative



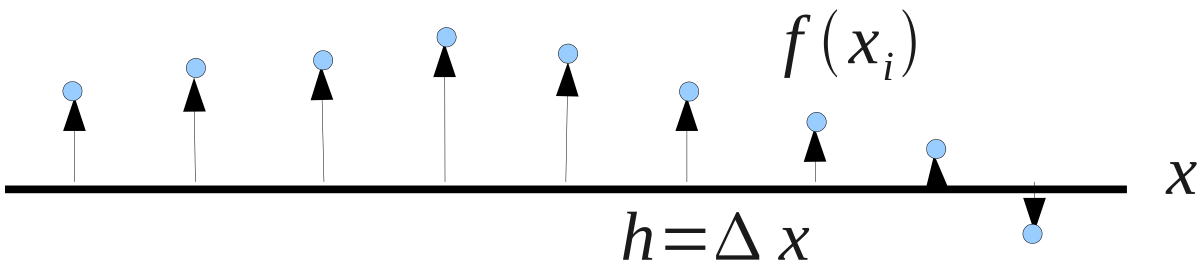
Both plots together



Demonstration

- Look at Matlab code in /home/sdb/Northeastern/Class2:
  - derivative.m
  - test_derivative.m
- To run it, do this:
  - [x, y] = test_derivative()

Derivatives as Matrix Multiplications



- f(x) is a vector of values evaluated at each x_i: $[f_0, f_1, f_2, f_3, \cdots]$
- Derivative (one-sided): $\frac{1}{h}[f_1 - f_0, f_2 - f_1, f_3 - f_2, \cdots]$

$$\frac{\partial f}{\partial x} = \frac{1}{h}\begin{pmatrix} -1 & 1 & 0 & 0 & \cdots \\ 0 & -1 & 1 & 0 & \cdots \\ 0 & 0 & -1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix}$$

Second derivative

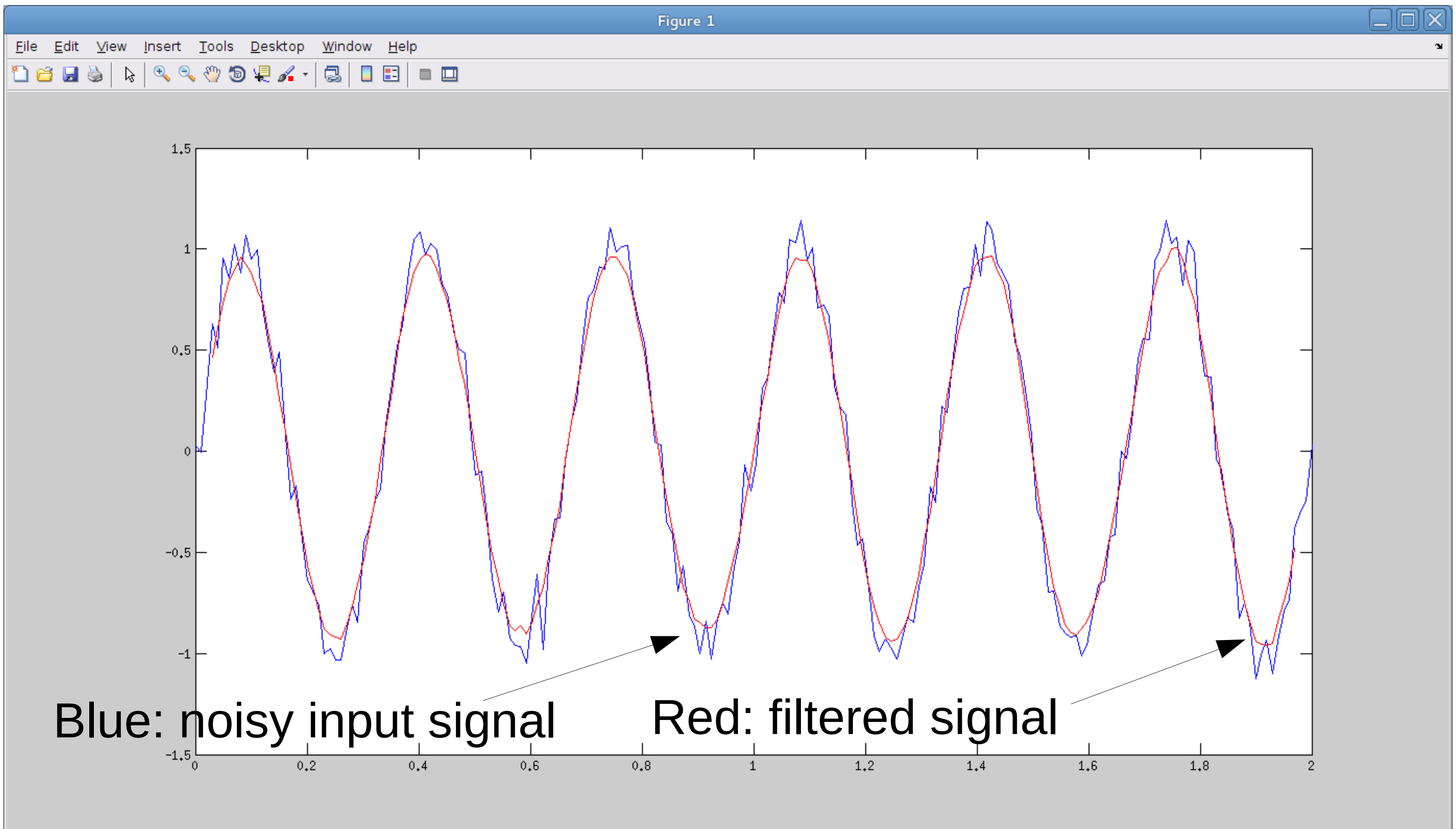
$$\frac{\partial^2 f}{\partial x^2} = \frac{1}{2h}\begin{pmatrix} -2 & 1 & 0 & 0 & \cdots \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix}$$

$$\frac{1}{2h}[f_1 - 2f_0, f_2 - 2f_1 + f_0, f_3 - 2f_2 + f_1, \cdots]$$

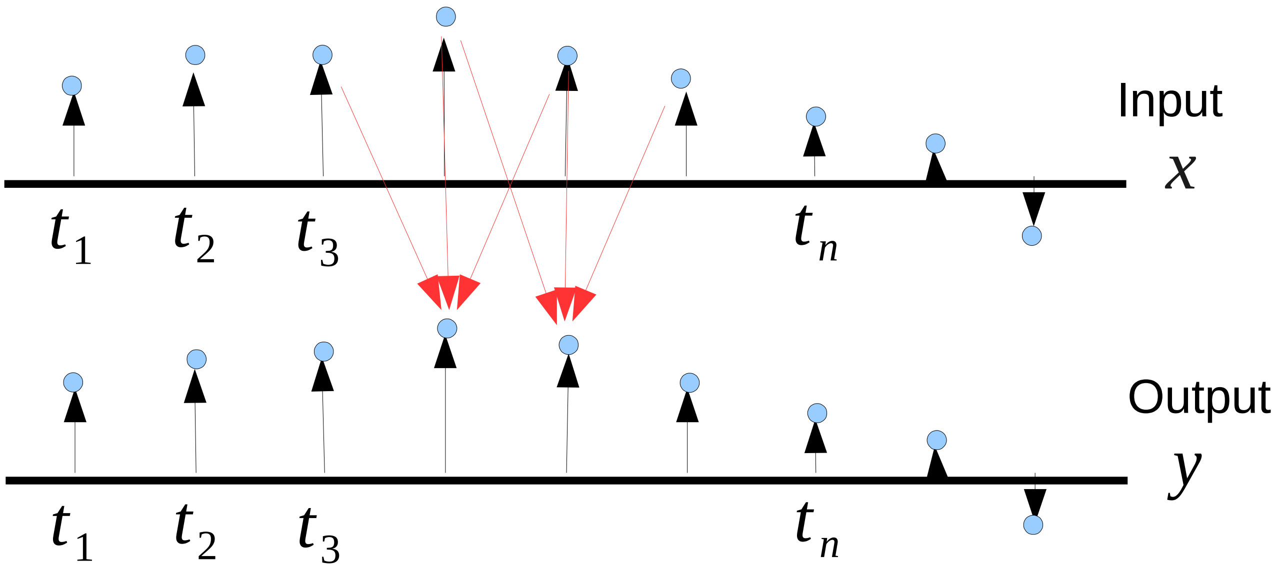
- We will see this again ....
- Note issue with boundary.
- A matrix is a linear operator.

Next topic: Filtering of data

- Input: Noisy data ← Imagine an audio stream (music)
- Desired output: Data with noise removed.



Simplest filter: moving box average



Three-point centered average.

- Idea: Take average of surrounding samples. Do this for each sample.

$$y_n = \frac{x_{n-1} + x_n + x_{n+1}}{3}$$

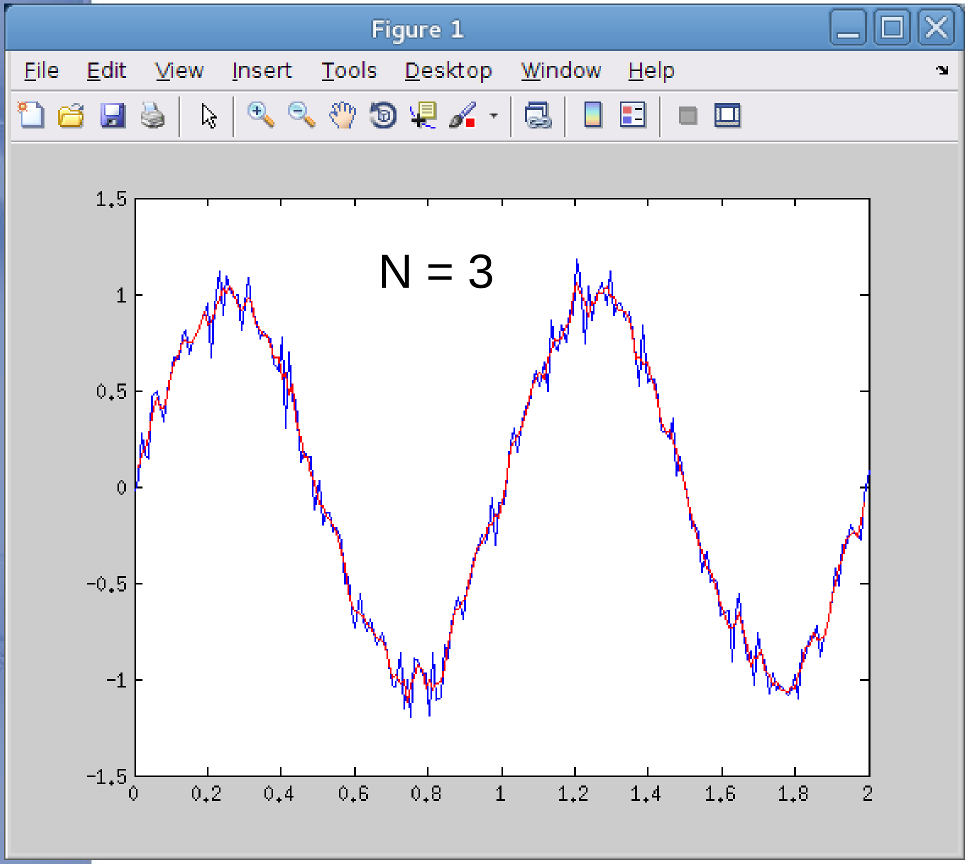
- The hardest part is getting the index arithmetic right....

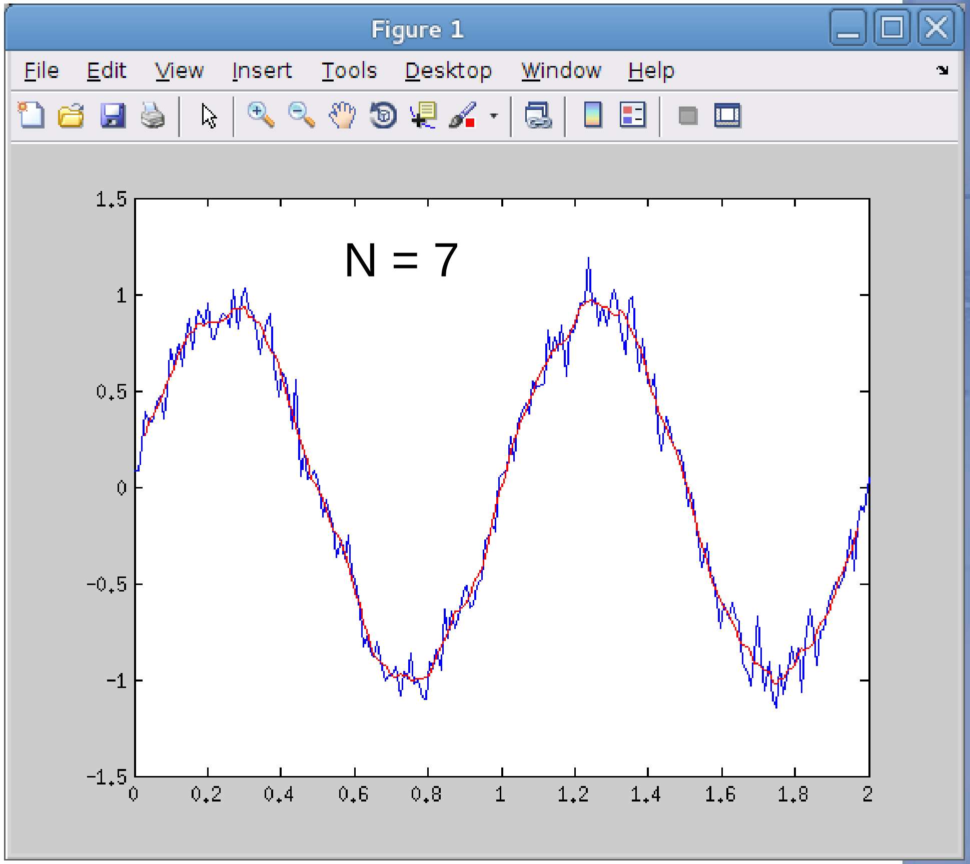
```
function [tf, yf] = box_filter_centered(t, x, Npts)
  % Performs centered box average over Npts points.

  % Check that Npts is an odd number
  if (mod(Npts, 2) == 0)
    error('Npts must be an odd number!')
  end

  % Compute number of points to the left & right
  Noffset = (Npts-1)/2;

  Nx = length(x)
  yf = zeros(Nx-Npts+1, 1);
  tf = zeros(Nx-Npts+1, 1);

  % Loop over input pts, compute box average
  for n = (1+Noffset):(Nx-Noffset)
    idx = (n-Noffset):(n+Noffset);
    tf(n-Noffset) = t(n);
    yf(n-Noffset) = sum(x(idx))/Npts;
  end

end
```

Note that I create a new time series vector along with the signal vector.

Here's where we compute the average of the input signal

Effect of different Npts





- Averaging more points together -> less noise in signal.

Some remarks

- The general computation is

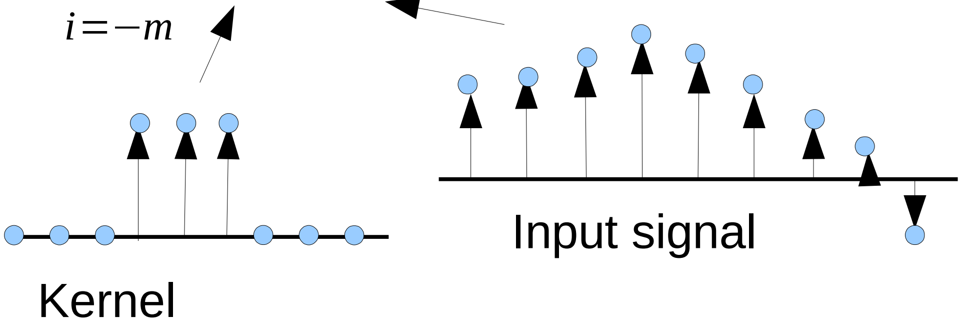
$$y_n = \sum_{i=-m}^{m} w_{n-i} x_i$$

Convolution –
Remember this
expression!

- Coefficients w_n must sum to 1 (to preserve "energy" in signal).
- How to deal with points at end?
- Concept: causal vs. non-causal filters
  - Centered average filter is non-causal.

Filter kernels

$$y_n = \sum_{i=-m}^{m} w_{n-i} x_i$$

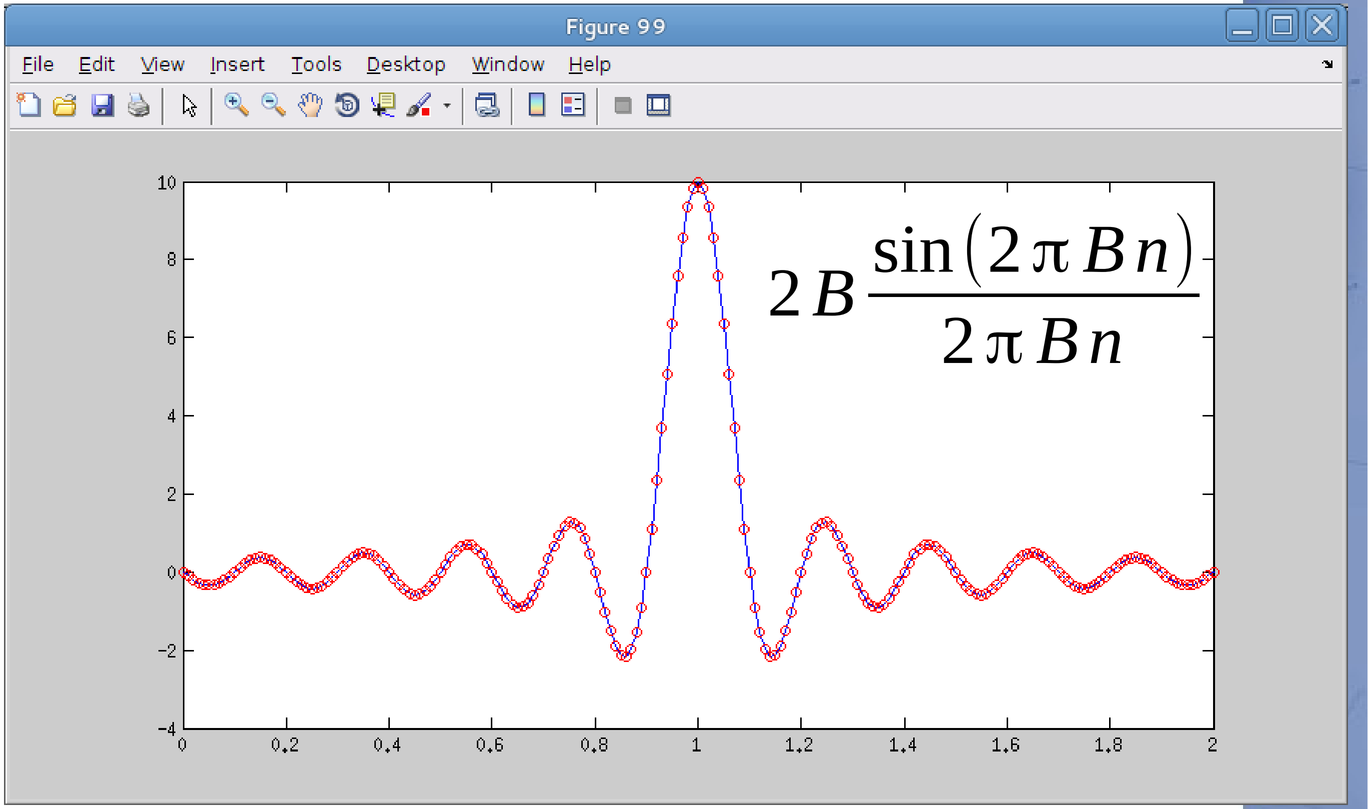


- Regard w coefficients as a function.
  - That function is called the “kernel”.
- Different kernels give different filter characteristics. (Recall effect of different Npts.)

Example: Sinc kernel

- Consider function sinc(x) = sin(x)/x
- Use as kernel in filter:

$$y_n = \sum_{i=-m}^{m} w_{n-i} x_i$$

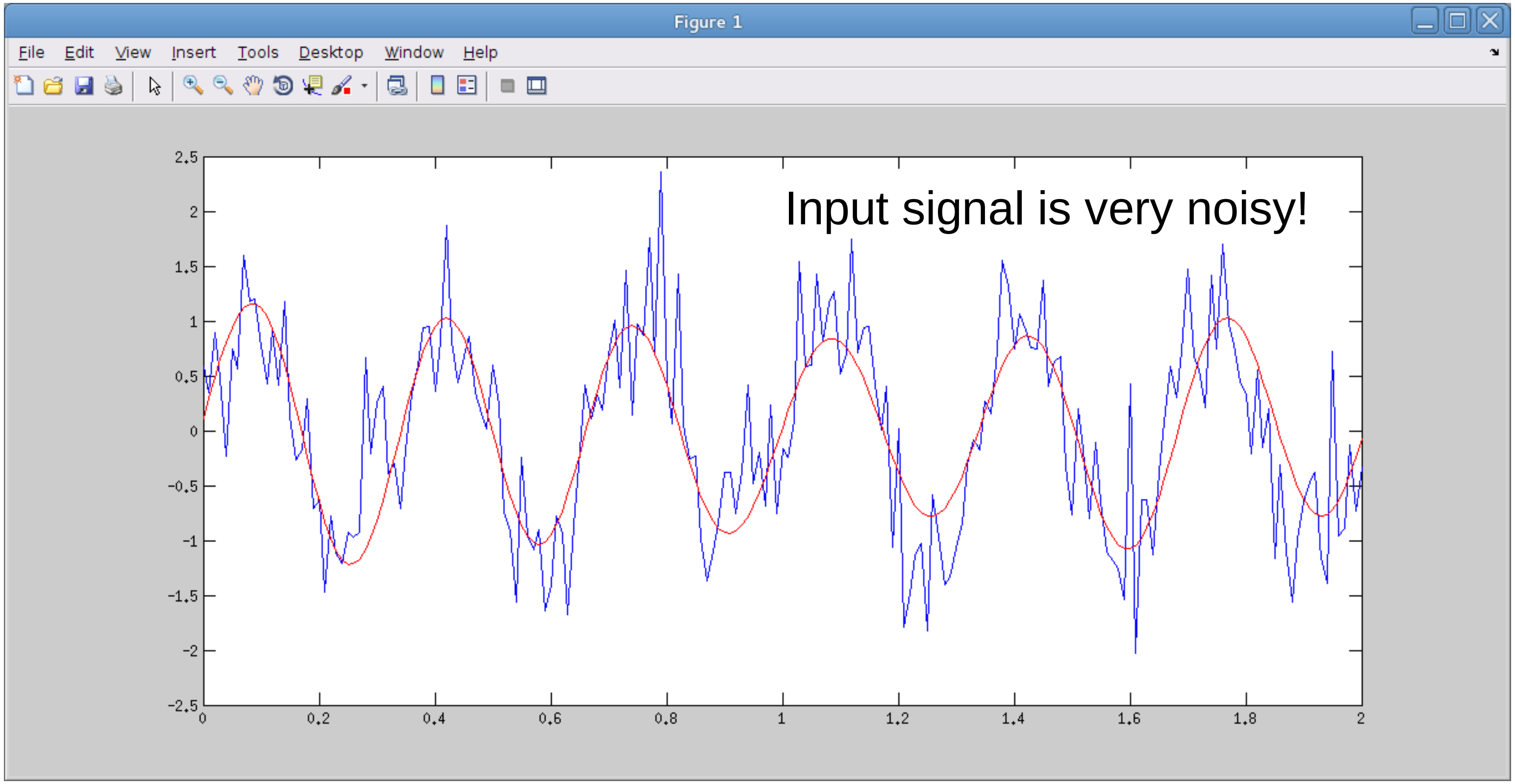


- Why use this crazy function?

To be revealed in session 2

```
function yf = sinc_filter_centered(t, x, B)
  % Filters x using sinc kernel.  The desired filter bandwidth
  % (cut-off freq) is B.  We apply the filter to a cyclic
  % version of the input signal.  That is, we assume the input
  % x(t) is periodic, and the input vector contains one period of x.

  % For everything to work, we require length(t) to be odd.
  N = length(t);
  if (mod(N, 2) == 0)
    error('length(t) must be odd!')
  end

  % Create filter kernel
  Tmax = t(end);
  w = 2*B*sinc(2*B*(t-Tmax/2));
  % Now shift it 1/2 around
  w = circshift(w, [0, (N-1)/2]);

  figure(99)
  plot(t, w);
  hold on
  plot(t, w, 'ro');

  % Create index used in computation
  idx = 1:N;

  % Loop over input pts, compute filtered signal
  for i = 1:N
    j = circshift(idx, [0, i-1]);
    yf(i) = dot(x(idx), w(j))/(N/2);
  end

end
```

Filtered signal

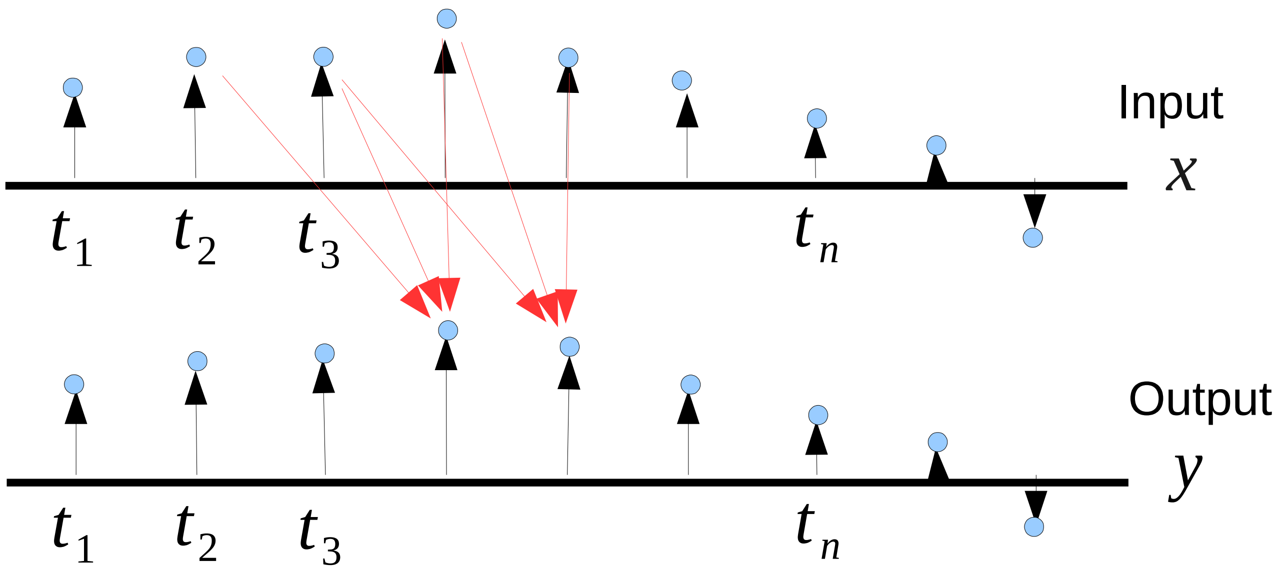


$f_0 = 3\,Hz \qquad B = 4\,Hz \qquad A_n = 0.5$

- Sinc() is applied to cyclic copies of input signal to deal with question of signal ends.
- Noise is very successfully reduced.

Causal filter (trailing box average)

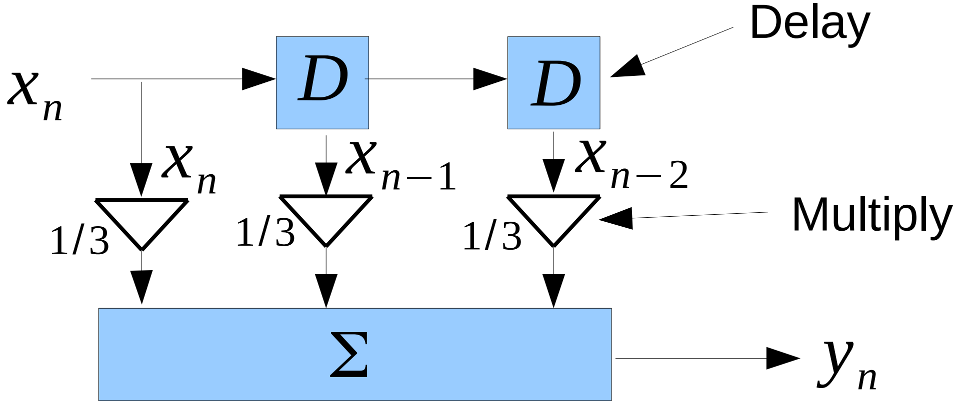
Three-point centered average.



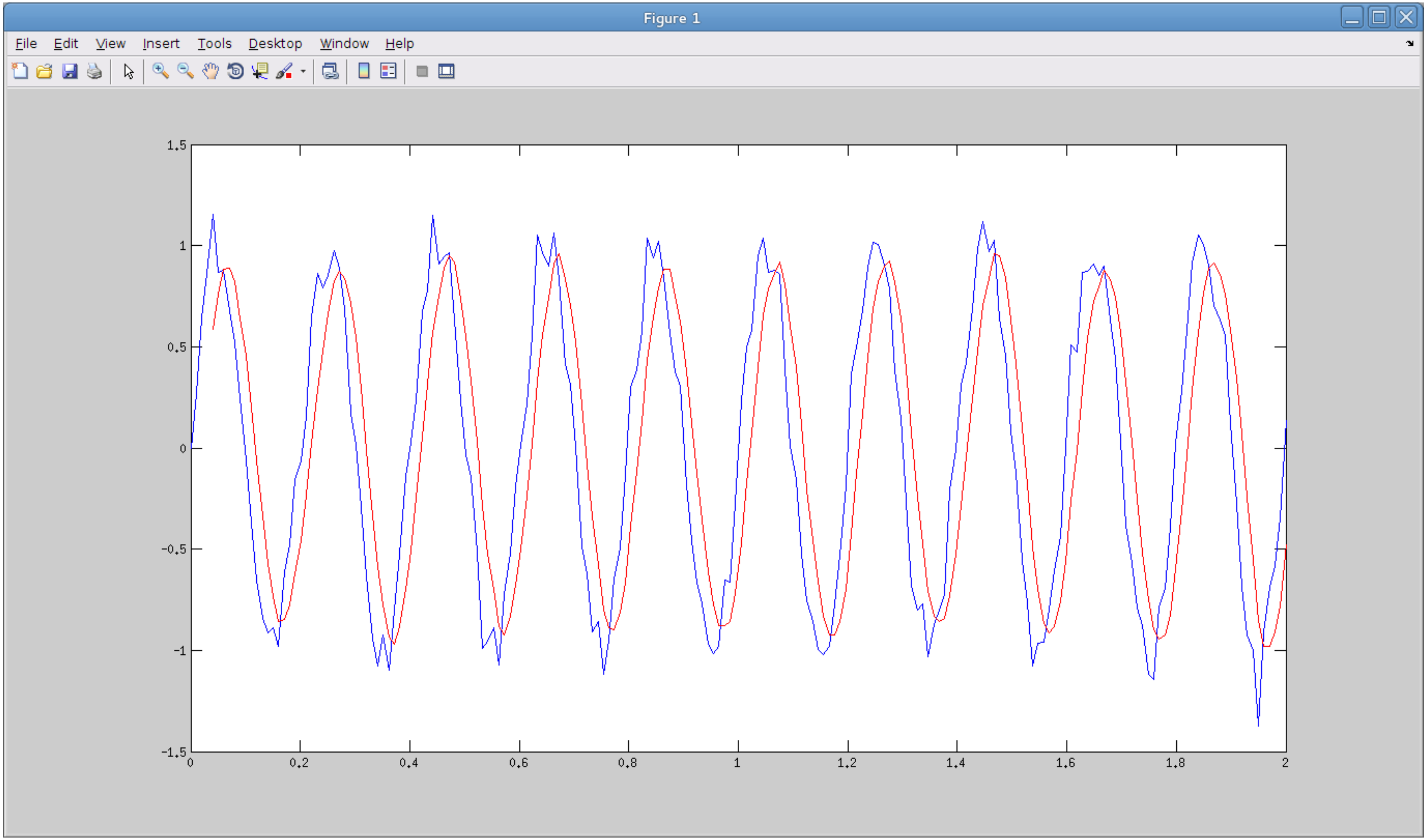
$$y_n = \frac{x_{n-2} + x_{n-1} + x_n}{3}$$

Note that this filter depends only upon present and past values of x (not upon future values).

Dataflow diagram



Filtering with trailing box average



- Note output signal has been delayed

Simple moving average

- Note SMA is delayed

Takeaway points

- You can filter a signal using a weighted moving average.
- The weights themselves can be considered to be a function
  - This function is the filter kernel
- Different kernels have different properties
- What kernel to use depends upon your specific signal and your specific goals.

Fourier series and Fourier transforms

Joseph Fourier

1768 -- 1830

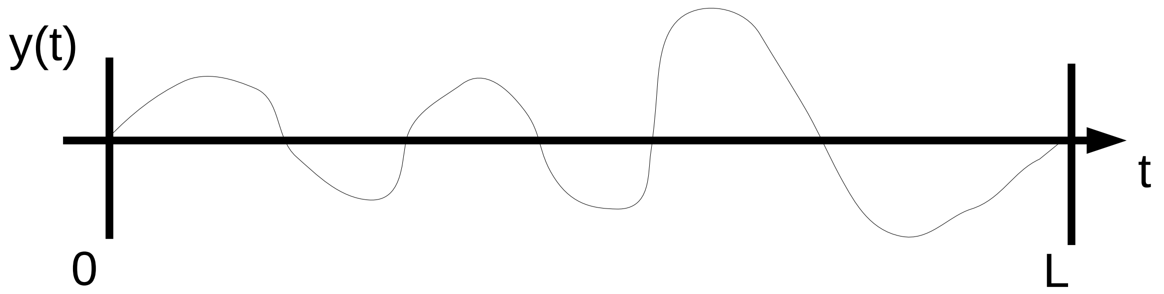
Goal: Expanding a function

- It's all about writing an expansion for a given function *y(t)*.
- Recall Taylor's series expansion around a point:

$$y(t-t_0)=a_0+a_1(t-t_0)+a_2(t-t_0)^2+a_3(t-t_0)^3+\cdots$$

- Properties:
  - Provides good approximation to $y(t-t_0)$ in neighborhood $t\approx t_0$
  - Usually only need a few terms for good approximation.

Consider function on finite interval



- Taylor expansion not very good here – polynomial order required is too high.
- Can I do a different expansion which works over entire interval?
- Note this fcn is zero at boundaries
- Yes: Fourier sin series:

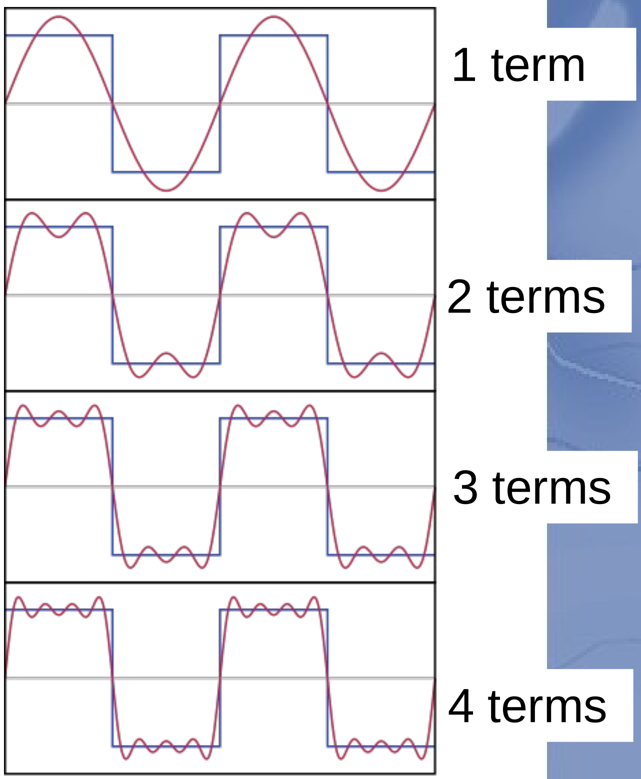
$$y(t) = a_1 \sin\left(\frac{\pi t}{L}\right) + a_2 \sin\left(\frac{2\pi t}{L}\right) + a_3 \sin\left(\frac{3\pi t}{L}\right) + \cdots$$

Fourier sin series

- Important theorem: I can expand any bounded, continuous function which is zero at the boundaries as a sum of sin functions (Fourier, 18th Century).

$$y(t)=\sum_{n=1}^{\infty} a_n \sin(n\pi t/L)$$

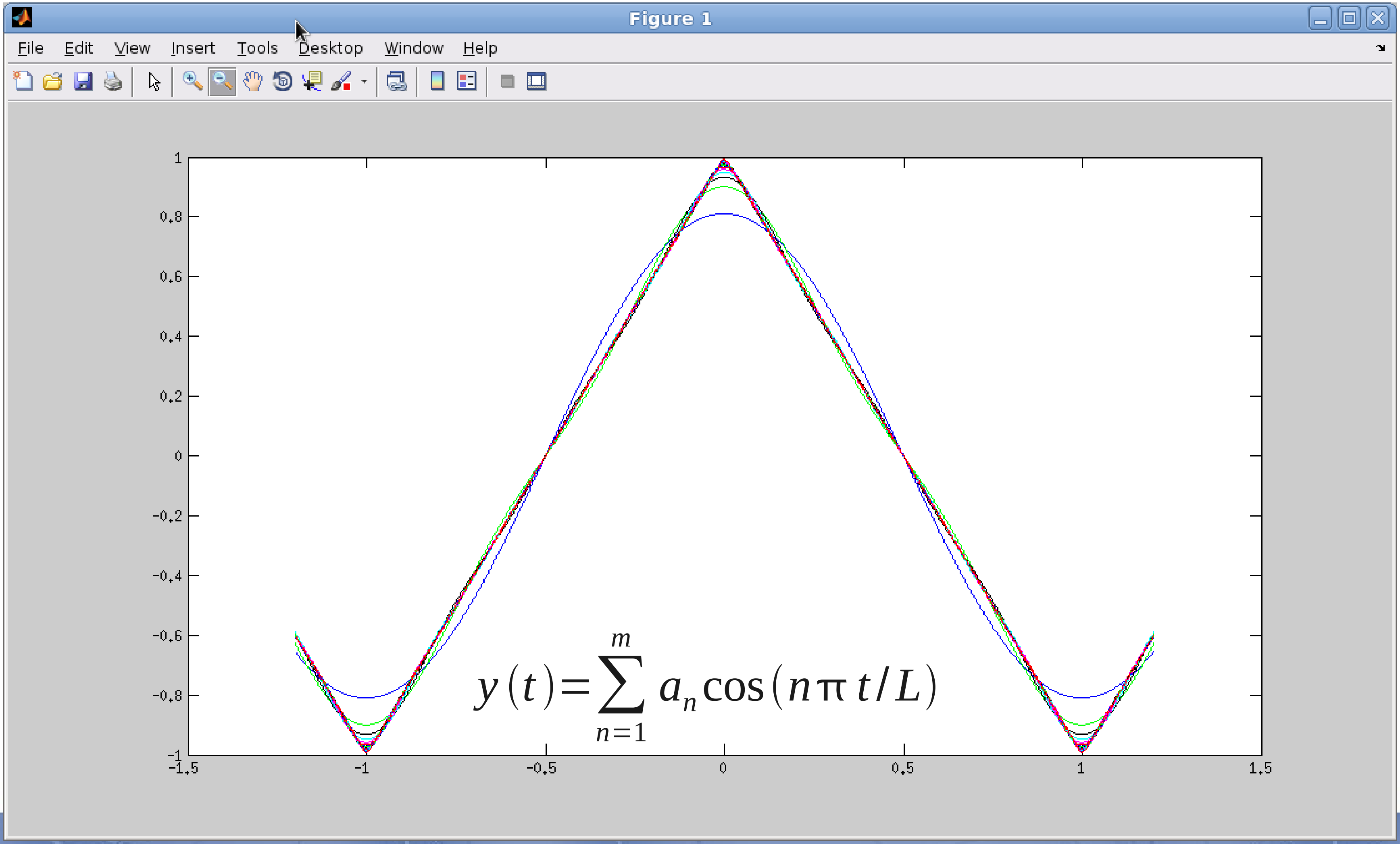
- I might need an infinite number of terms.
- Series converges over entire interval.
- Each term in expansion has coefficient a_n
- But how to get coefficients?



There is a similar theorem involving cos(t)

Fourier series is remarkable

- Says you can expand even functions with discontinuities using sin/cos functions.



How to get coefficients?

- Consider integrating the product of two sin functions:

$$\int_0^L dt \sin\left(\frac{n\pi t}{L}\right)\sin\left(\frac{m\pi t}{L}\right)$$

m, n are integers

- Recall trig identity:

$$\sin(x)\sin(y)=\frac{1}{2}(\cos(x-y)-\cos(x+y))$$

- So: $$\int_0^L dt \sin\left(\frac{n\pi t}{L}\right)\sin\left(\frac{m\pi t}{L}\right)$$

$$=\frac{1}{2}\int_0^L dt\left[\cos\left(\frac{(n-m)\pi t}{L}\right)-\cos\left(\frac{(n+m)\pi t}{L}\right)\right]$$

Deriving coefficients.....

- Consider expression

$$\frac{1}{2}\int_0^L dt\left[\cos\left(\frac{(n-m)\pi t}{L}\right)-\cos\left(\frac{(n+m)\pi t}{L}\right)\right]$$

- When $n\neq m$ we have two terms like:

$$\int_0^L dt\cos\left(\frac{p\pi t}{L}\right)$$

p is integer

$$=\frac{L}{p\pi}\sin\left(\frac{p\pi t}{L}\right)\Bigg|_0^L$$

Draw picture on blackboard to show why this integrates to zero

0

$$=\frac{L}{p\pi}\left(\sin p\pi-0\right)=0$$

When n = m...

$$\frac{1}{2}\int_0^L dt\left[\cos\left(\frac{(n-m)\pi t}{L}\right)-\cancelto{=0}{\cos\left(\frac{(n+m)\pi t}{L}\right)}\right]$$

$$=\frac{1}{2}\int_0^L dt\cos\left(\frac{0\pi t}{L}\right)$$

$$=\frac{L}{2}$$

- Conclusion: integral is non-zero only when n = m.

Orthogonal functions

- Sin functions are orthogonal over interval [0, L]

$$\int_0^L dt \sin\left(\frac{n\pi t}{L}\right)\sin\left(\frac{m\pi t}{L}\right) \begin{cases} =0 \text{ for } n\neq m \\ =\dfrac{L}{2} \text{ for } n=m \end{cases}$$

- Similar to orthogonality of vectors:

$$\vec{u}\cdot\vec{v} \quad \begin{matrix} =0 \text{ for } \vec{u}\neq\vec{v} \\ =C \text{ for } \vec{u}=\vec{v} \end{matrix}$$

Consider what this means for Fourier expansion

- Start with

$$y(t)=\sum_{n=1}^{\infty} a_n \sin(n\pi t/L)$$

- Multiply through both sides and integrate:

$$\int_0^L dt\, y(t)\sin\left(\frac{m\pi t}{L}\right)=\sum_{n=1}^{\infty} a_n \int_0^L dt\,\sin\left(\frac{m\pi t}{L}\right)\sin\left(\frac{n\pi t}{L}\right)$$

- Use orthogonality:

Method to get coefficients

$$\boxed{a_m=\frac{2}{L}\int_0^L dt\, y(t)\sin\left(\frac{m\pi t}{L}\right)}$$

Therefore, we can go in two directions

- Fourier series expansion:

$$y(t) \Leftrightarrow \sum_{n=1}^{\infty} a_n \sin(n\pi t/L)$$

- You can go back and forth:

$$y(t) = \sum_{n=1}^{\infty} a_n \sin(n\pi t/L)$$

Get function from coefficients

$$a_m = \frac{2}{L} \int_0^L dt\, y(t) \sin\left(\frac{m\pi t}{L}\right)$$

Get coefficients from function

Generalize to any function defined on an interval

- You can expand any function, regardless of values on boundary using:

Real *y(t)*

$$y(t)=\frac{a_0}{2}+\sum_{n=1}^{\infty} a_n \cos(\omega_n t)+\sum_{n=1}^{\infty} b_n \sin(\omega_n t)$$

a, b are real

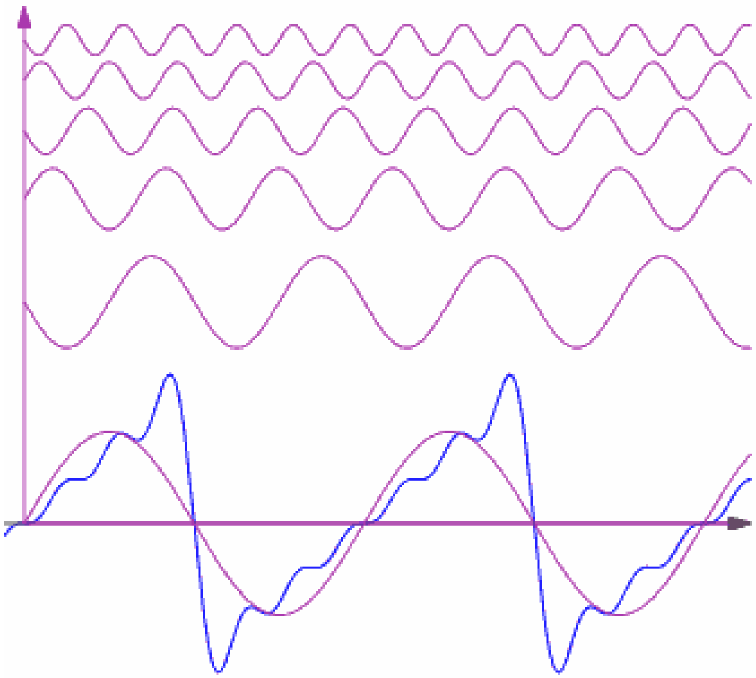
Complex *y(t)*

$$y(t)=\sum_{n=-\infty}^{\infty} c_n e^{-i\omega_n t}$$

c is complex

Consider meaning of a_n coefficients

- Define $\omega_n = n\pi / L$
- Then Fourier series is $y(t) = \sum_{n=1}^{\infty} a_n \sin(\omega_n t)$
- Interpret ω_n as a frequency
- Height of a_n determines amplitude of that frequency component in signal *y(t)*.



- **Key point:** Any signal can be viewed as composed of a sum of sin/cos waves.

MIT Mathlets

- Demo: http://mathlets.org/mathlets/fourier-coefficients/
- To generate square waves, coefficients are:

$$a_n = \frac{4}{n\pi} \quad \text{Odd n}$$

$$= 0 \quad \text{Even n}$$

```
>> n = 1:2:11

n =

    1     3     5     7     9    11

>> an = (4./(n*pi))'

an =

   1.27323954473516
   0.424413181578388
   0.254647908947033
   0.181891363533595
   0.141471060526129
   0.115749049521378
```

Complex Fourier series

- Fourier series expansion:

$$y(t) = \sum_{n=-\infty}^{\infty} a_n e^{-int}$$

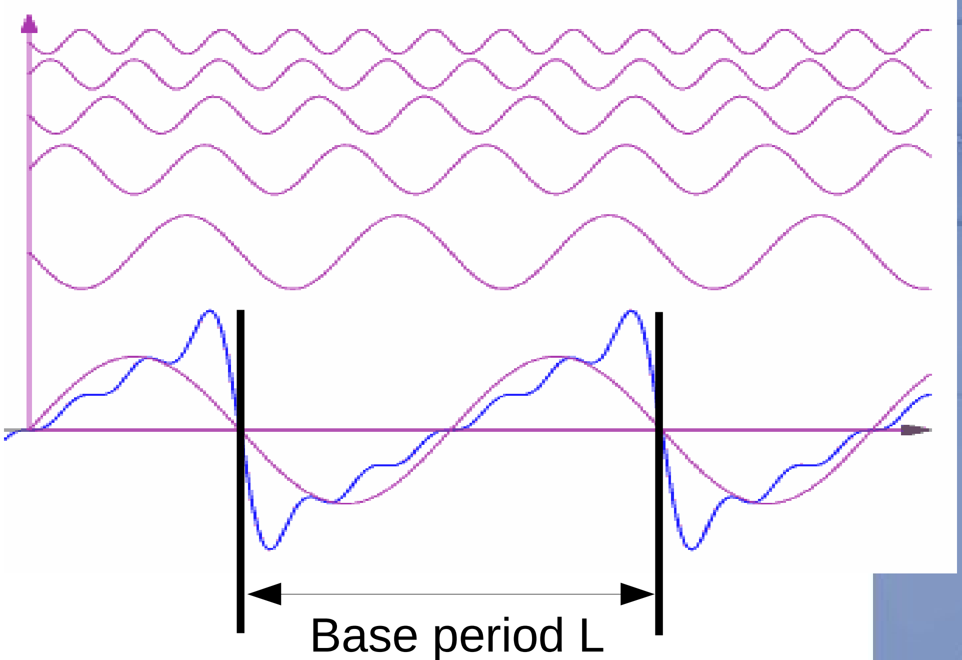
- Given coefficients, compute function:

$$a_m = \frac{1}{2\pi} \int_{-\pi}^{\pi} dt\, y(t) e^{int}$$

- Note different limits of summation and integration.

What happens outside interval [0, L]?

- Basis functions sin, cos mean expansion extends to infinity, and is periodic.
- Base period L
- Therefore, you can use Fourier series to expand:
  - Any periodic function
  - Any function defined on a finite interval



Fourier transform

- Fourier series defined for signal on finite interval or periodic.
- What if signal is infinite (i.e. extends to $t=\pm\infty$)?
- Fourier transform pair:

Transform to frequency domain

$$Y(\omega)=\int_{-\infty}^{\infty} dt\, y(t)\, e^{-i\omega t}$$

Transform to time domain

$$y(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty} d\omega\, Y(\omega)\, e^{i\omega t}$$

Fourier transform vs. series

Fourier series

$$a_m = \frac{2}{L}\int_0^L dt\, y(t)\sin\left(\frac{m\pi t}{L}\right)$$

$$y(t) = \sum_{n=1}^{\infty} a_n \sin\left(\frac{n\pi t}{L}\right)$$

- Valid for:
  - Periodic function
  - Function on interval
- Continuous function, discrete spectrum

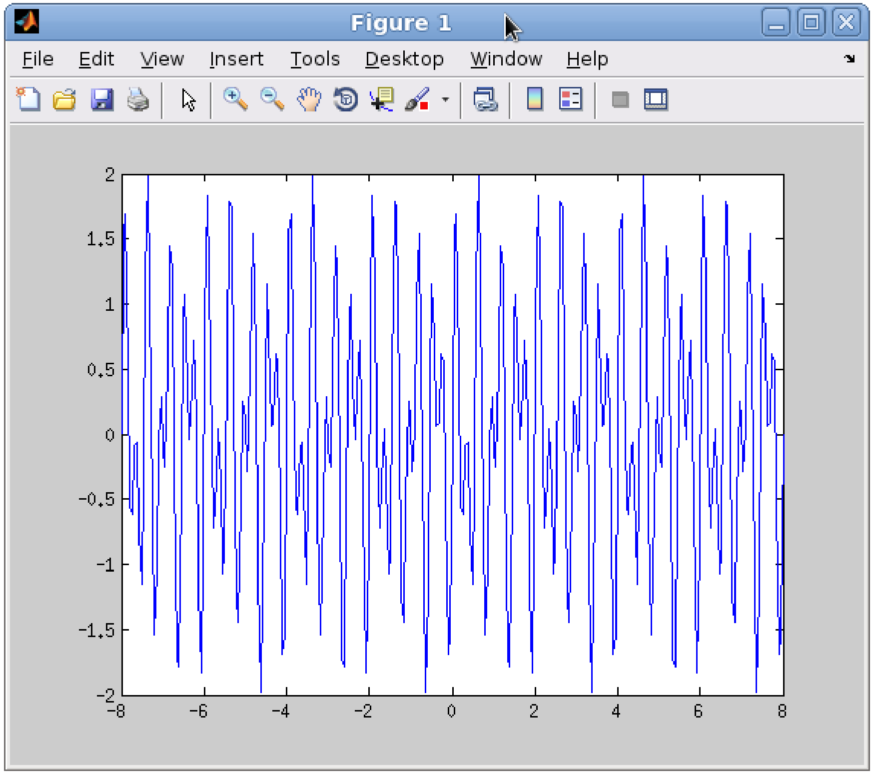
Fourier transform

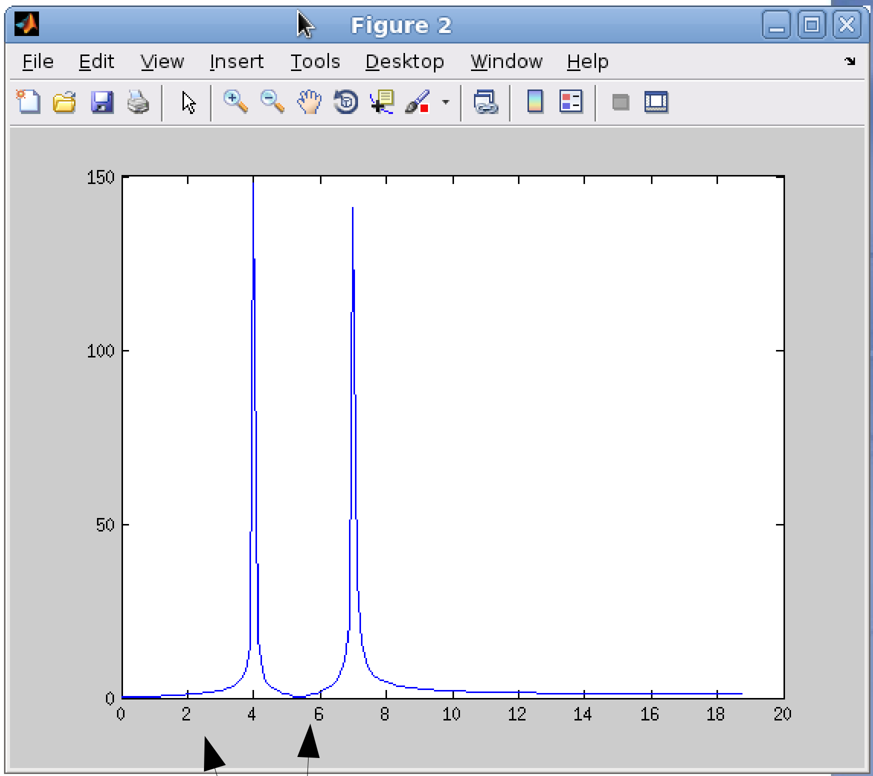
$$Y(\omega) = \int_{-\infty}^{\infty} dt\, y(t)\, e^{-i\omega t}$$

$$y(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} d\omega\, Y(\omega)\, e^{i\omega t}$$

- Valid for any function
- Function and spectrum continuous.

Fourier transform converts time-varying signal into frequency spectrum






```
f1 = 4;
f2 = 7;
w = -t.*t + 64;
y = w.*(sin(2*pi*f1*t) + sin(2*pi*f2*t));
```

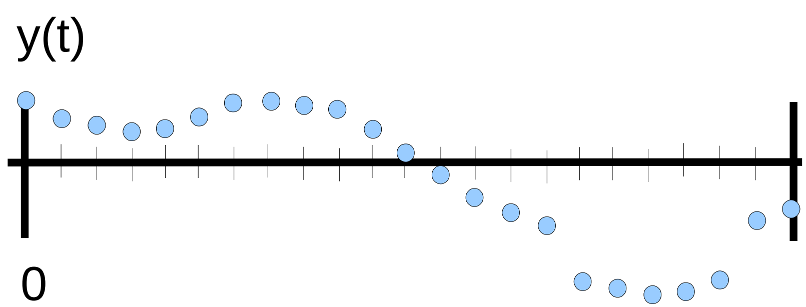
Create signal with two frequencies:
4Hz and 7Hz.

Take Fourier transform.
Observe two delta functions
at 4 and 7 Hz.

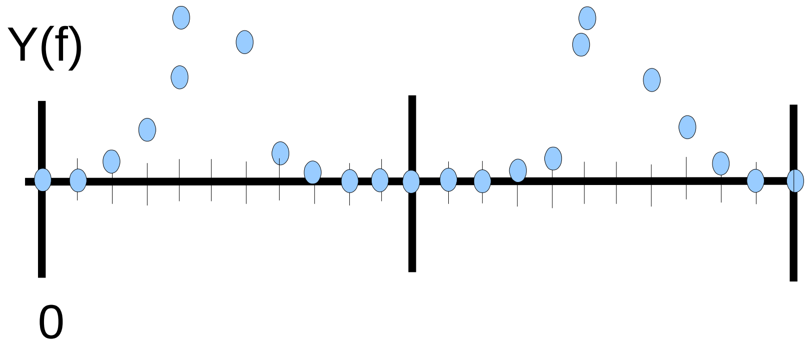
Time and frequency are duals

- Fourier transform pair: you can go back and forth from time domain to frequency domain.

$$y(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty} d\omega\, Y(\omega)\, e^{i\omega t} \quad \longleftrightarrow \quad Y(\omega)=\int_{-\infty}^{\infty} dt\, y(t)\, e^{-i\omega t}$$

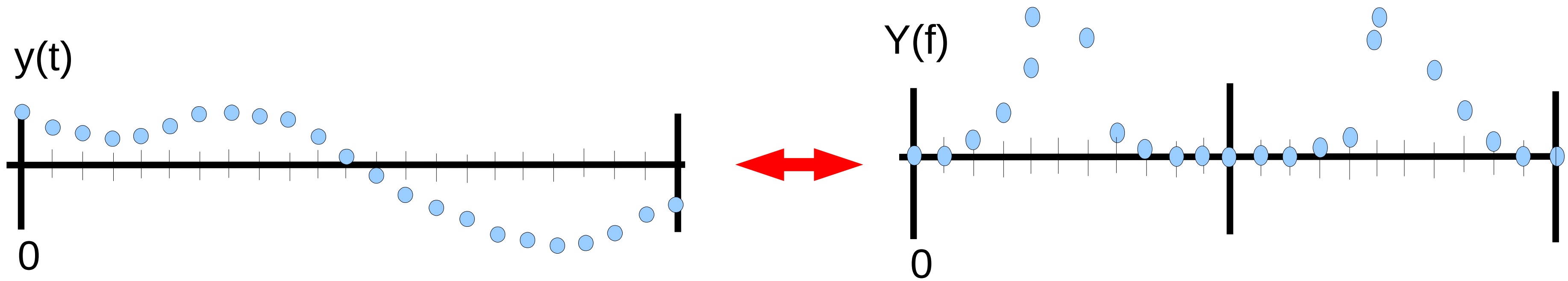


Time domain



Frequency domain

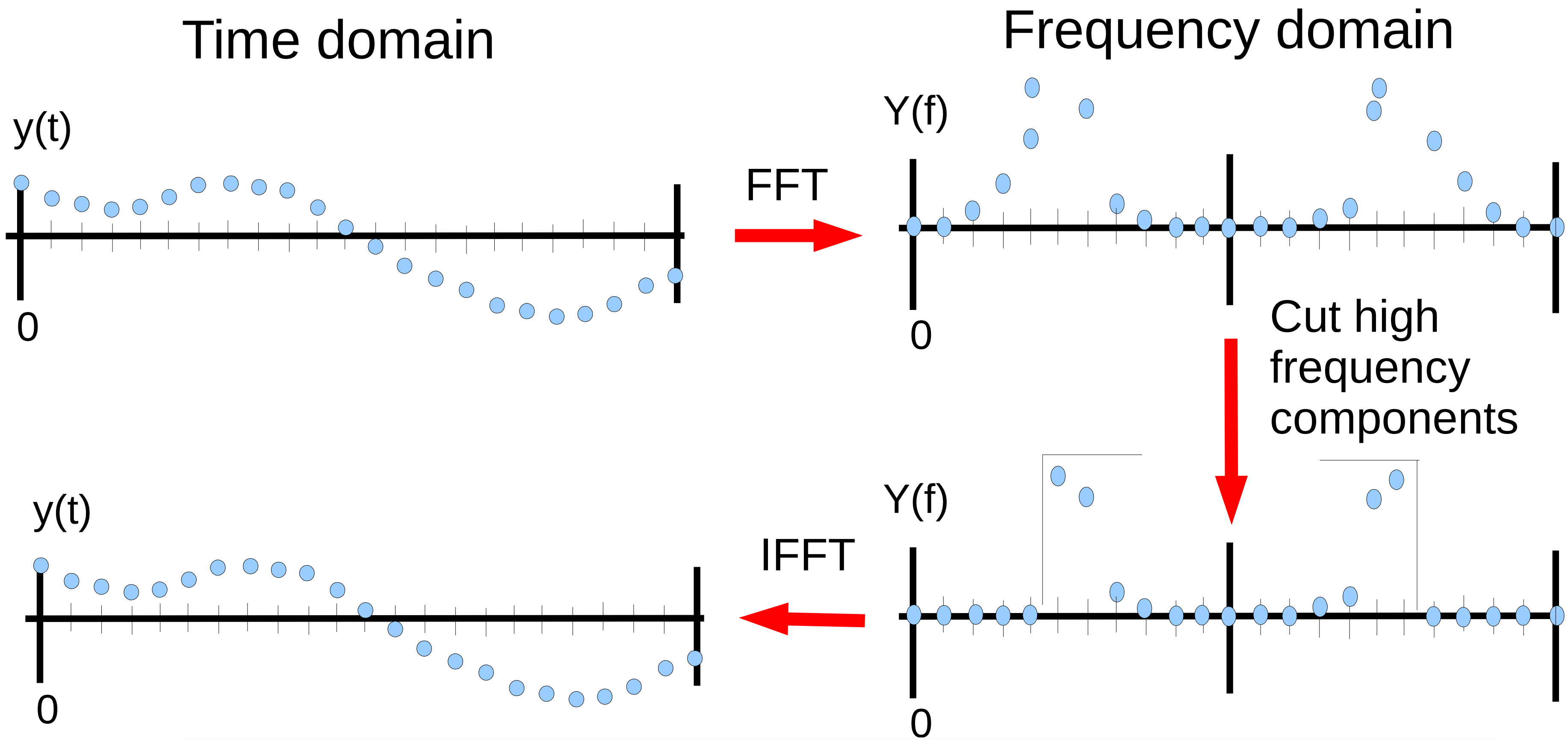
Time domain and frequency domain

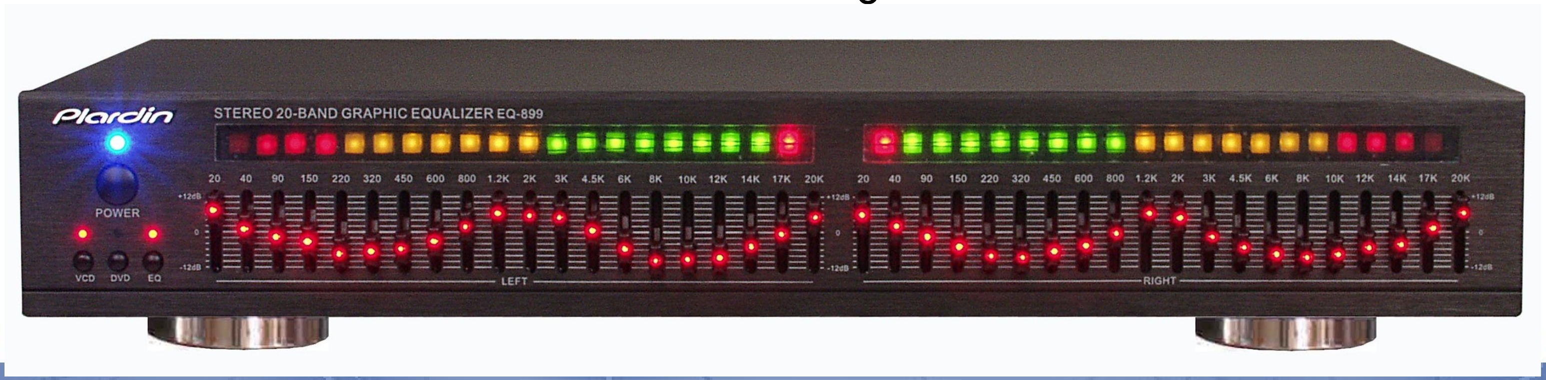


Time domain

Frequency domain

Application: filtering





Stereo equalizer

Session summary

- Sampled data
- Simple filters
- Taking numeric derivatives
- Fourier series
- Fourier transform