Computational Methods (PHYS 2030)

Instructors: Prof. Christopher Bergevin (cberge@yorku.ca)

Schedule: Lecture: MWF 11:30-12:30 (CLH M)

Website: http://www.yorku.ca/cberge/2030W2018.html

York University
Winter 2018
Lecture 16

Aside: ”AI” & Computing

The Economist February 17th 2018

Artificial intelligence

Peering into the black box

THERE is an old joke among pilots that says the ideal flight crew is a computer, a pilot and a dog. The computer's job is to fly the plane. The pilot is there to feed the dog. And the dog's job is to bite the pilot if he tries to touch the computer.

Aside: "AI" & Computing

The Economist February 17th 2018

Artificial intelligence

Peering into the black box

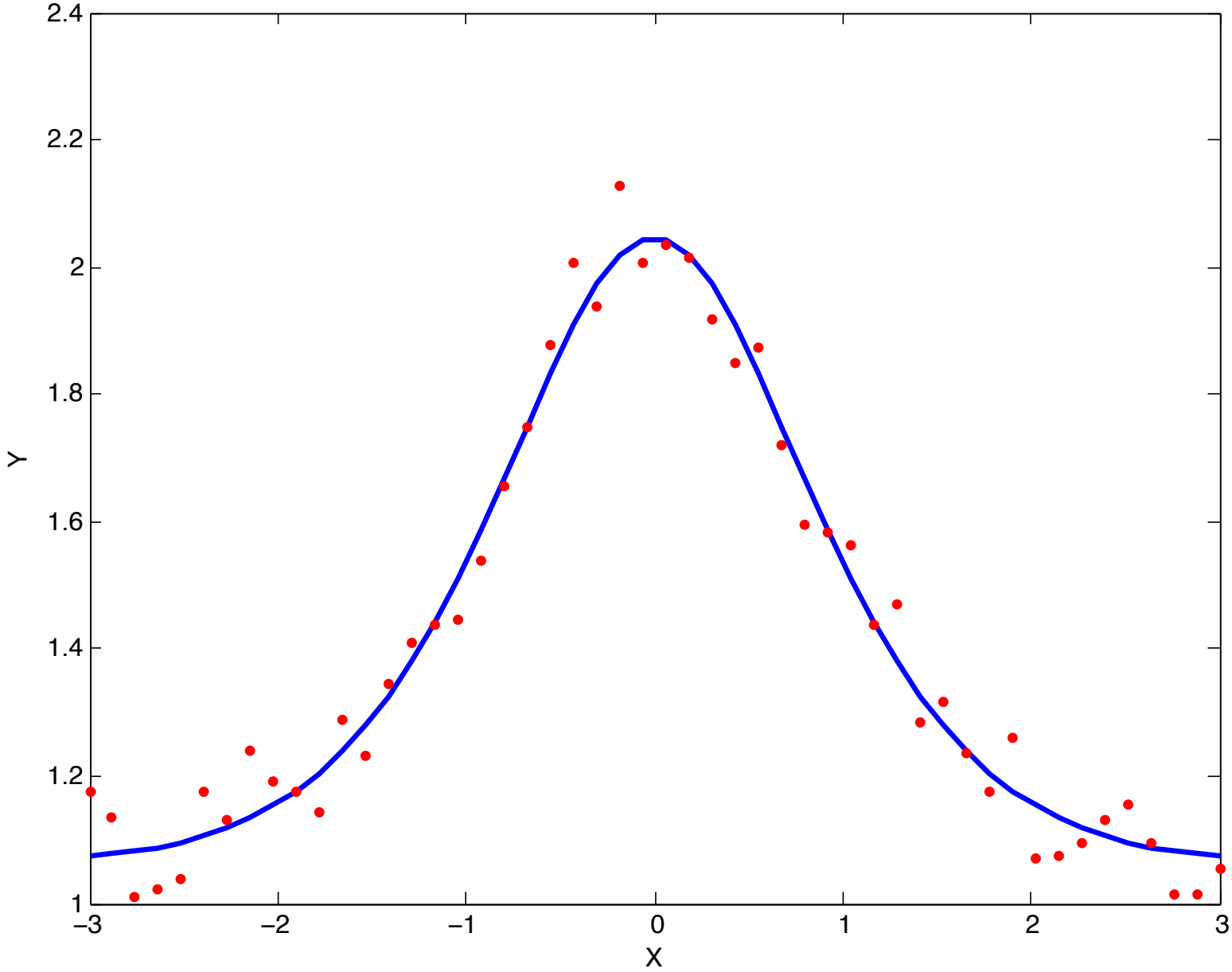
Handing complicated tasks to computers is not new. But a recent spurt of progress in machine learning, a subfield of artificial intelligence (AI), has enabled computers to tackle many problems which were previously beyond them. The result has been an AI boom, with computers moving into everything from medical diagnosis and insurance to self-driving cars.

PHYS 2030 is chiefly focused on specifically telling a computer what to do.
But the "future" of computing is likely a bit different:

Tell the computer how to "learn" so to do it itself….

Review: Nonlinear regression

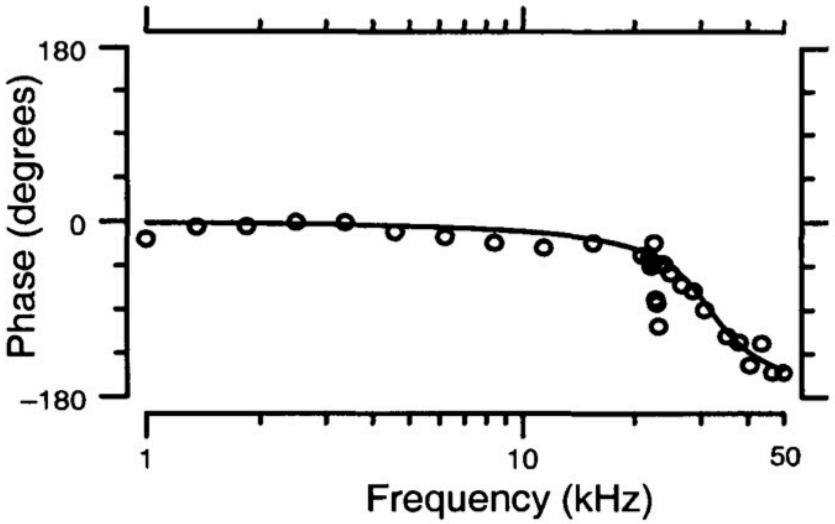
EXregression5.m



Nonlinear Regression

- What if you decide the data is not best fit with a simple linear function nor a polynomial?

(sinusoidally driven) Micro-mechanical resonator steady-state phase



$$\delta(\omega) = \arctan\left(\frac{\gamma\omega}{\omega^2 - \omega_o^2}\right)$$

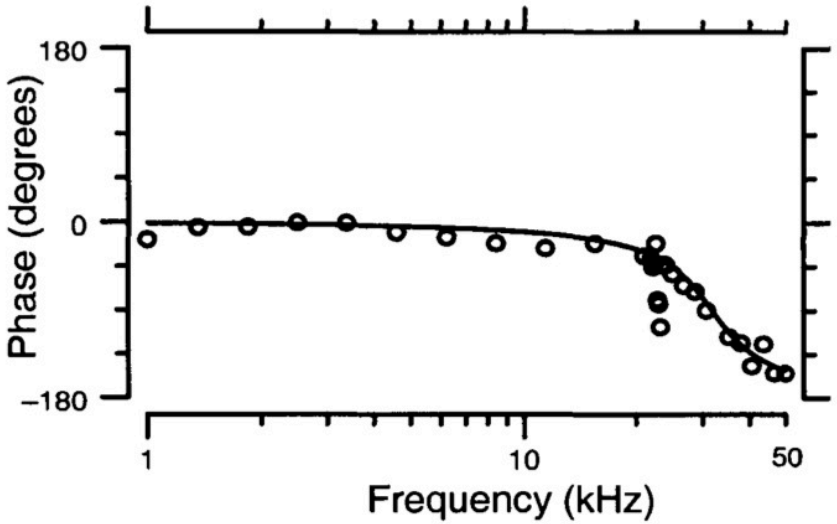
→ Let's try to fit the (nonlinear) function we have as our "model"

```
% User Inputs
fileL= 'extractedData.txt'; % file containing the steady-state phase data to fit

%Next, we'll define the initial parameters for the fit.
%fit.m requires the fitting function to have all possible model parameters
%to be fields of a single structure.  These fields can be single elements,
%or they can be vectors or matrices.
initP.AA = 1;   % provide some rough starting point for each parameters
initP.BB = 4;   % (only need to specify params. here relevant to fitting)

% 'freeList' is a cell array of strings that holds the names of the model
% parameters to be allowed to vary in the optimization routine.
%freeList = {'BB'}  % only fit one parameter, keep others const. at initial guess
freeList = {'AA','BB'};    % fit all three params.
% ------------------------------------------------

ssP= load(fileL);   % load in data and extract appropriate x and y values
xL= ssP(:,1);
xL= log(xL);    % do fit on log axes
y= ssP(:,2)*pi/180; % convert phase to rads

%Now plot the data and save the graphics handles.
figure(1); clf
%plot_handle = plot(xL,y,'b-','LineWidth',2);    hold on
plot_handle = semilogx(xL,y,'b-','LineWidth',2);    hold on
plot(xL,y,'r.');
text_handle = text(mean(xL),max(y)+1,'','HorizontalAlignment','center','FontSize',14);
xlabel('Frequency [kHz]');ylabel('Phase [rads]');

%Call 'fit'.  Note the order of the input parameters:
%1) fitted function name
%2) initial parameter structure
%3) list of free parameters
%4) 2nd parameter to be sent into the fitted function ...
[bestP,err] = fit('myTestFunction2',initP,freeList,xL,y,plot_handle,text_handle);
```

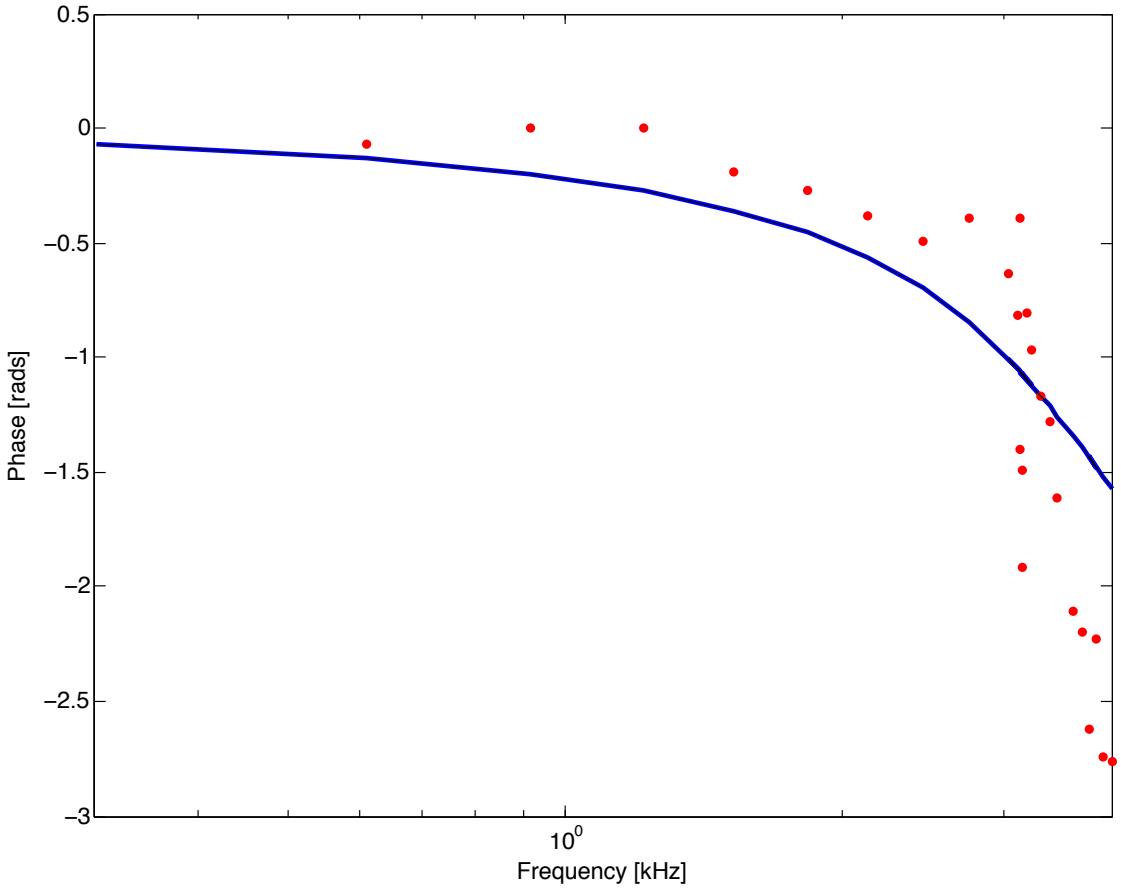
→ Remember that we extracted the values via deplot.m

```
function err = myTestFunction2(p,x,y,plot_handle,text_handle)
% Calculates the sum of squared error between y and the predicted function,
% which is the sum of a sinusoid and a 2nd order polynomial.

predY = atan((p.AA*x)./(x.^2-p.BB^2));   % explicit form of fitting function

% calculate chi-squared (this is what we want to minimize) by comparing to
% actual dependent variable (i.e. the y-data to fit to)
err = sum( (y-predY).^2);

% deal with graphics update
set(plot_handle,'YData',predY);
txt=sprintf(' ');
set(text_handle,'String',txt);
drawnow
```

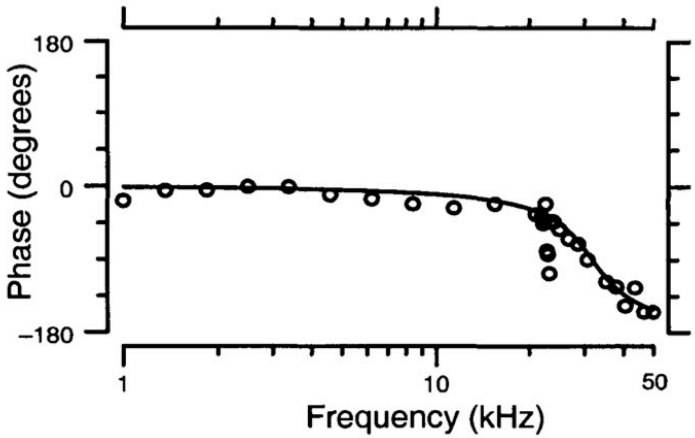
$$\delta(\omega) = \arctan\left(\frac{\gamma\omega}{\omega^2 - \omega_o^2}\right)$$



```
function [params,err] = fit(funName,params,freeList,varargin)
%Helpful interface to matlab's 'fminsearch' function.
%
%turn free parameters in to 'var'
if isfield(params,'options')
  options = params.options;
else
  options = [];
end

if isempty(freeList)
  freeList = fieldnames(params);
end

vars = params2var(params,freeList);

if ~isfield(params,'shutup')
  disp(sprintf('Fitting "%s" with %d free parameters.',funName,length(vars)));
end

vars = fminsearch('fitFunction',vars,options,funName,params,freeList,varargin);

%get final parameters
params=  var2params(vars,params,freeList);

%evaluate the function

evalStr = sprintf('err = %s(params',funName);
for i=1:length(varargin)
  evalStr= [evalStr,',varargin{',num2str(i),'}'];
end
evalStr = [evalStr,');'];
eval(evalStr);
```



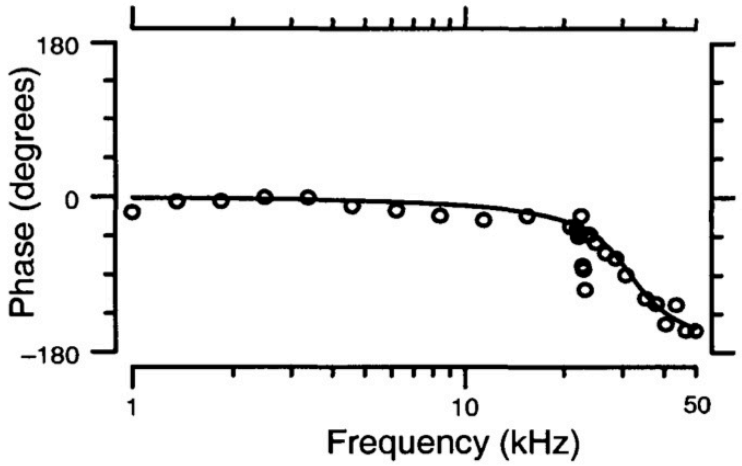
→ Why the worse fit?



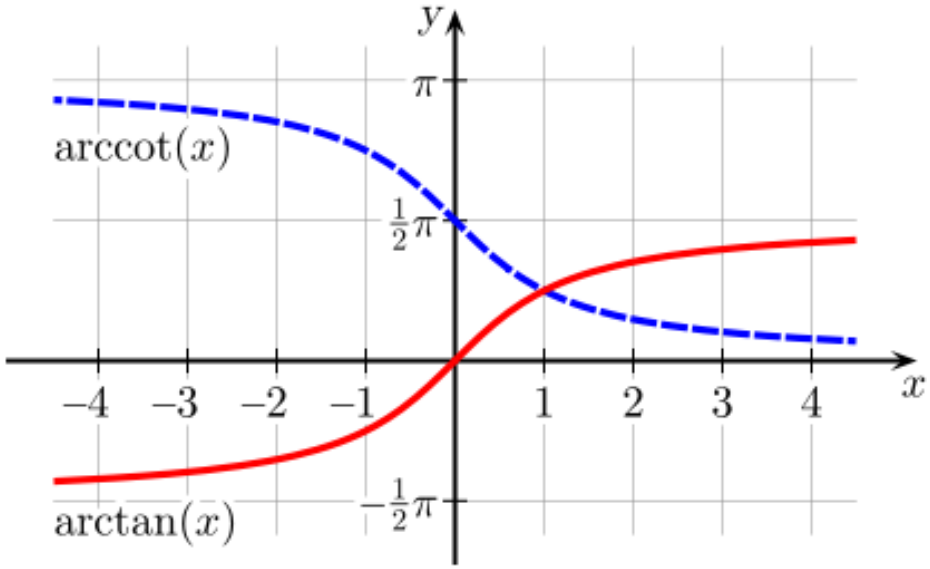
‘Golden rules’ of (2030) computing

$$\delta(\omega) = \arctan\left(\frac{\gamma\omega}{\omega^2 - \omega_o^2}\right)$$

#1 – **The computer only does what you tell it to do**



- Think about the nature of both the function you are trying to fit and the data



- Also consider:
  - Log abscissa
  - Extracted points match?
  - Better choice for ‘suggested’ initial parameters to try?

→ Think about how we could improve our calculation/code

wikipedia (inverse trigonometric functions)

To try yourself….

- Reproduce the determined fit (quadratic) parameters for the thermocouple example

- Compare how parameters differ for the intercept and slope when fitting a (noisy) set of linear data with a quadratic function

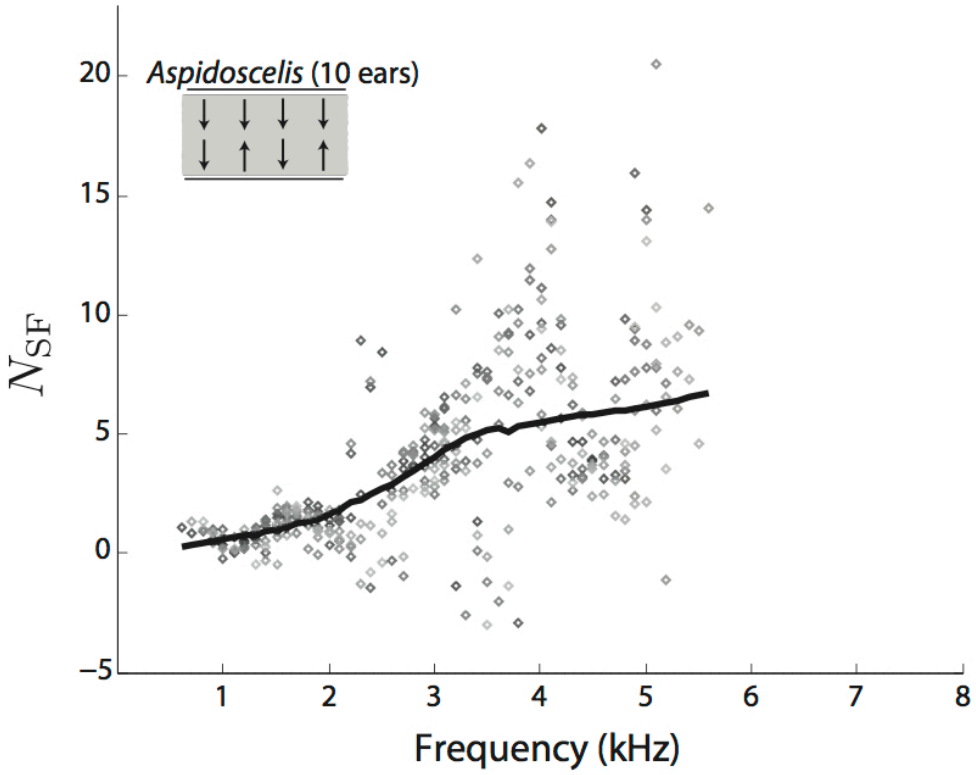
- Try running EXregression5.m. What happens if you specify a different ‘test function’?

- How might you modify EXregression6.m (or myTestFunction2.m) such that the fit to the micro-mechanical resonator steady-state phase is improved?

Nonparametric Regression

- What if you have no idea the form of the function to fit? Or you don't want to assume a particular form (i.e., you'd prefer to let the data dictate the trend)?

"Nonparametric regression is a form of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data. Nonparametric regression requires larger sample sizes than regression based on parametric models because the data must supply the model structure as well as the model estimates."



- What 'shape' do the data have? Is there a trend?
- How might we go about finding the 'best' trend with the fewest number of assumptions?

→ This sort of situation is very common in numerous scientific and engineering applications!

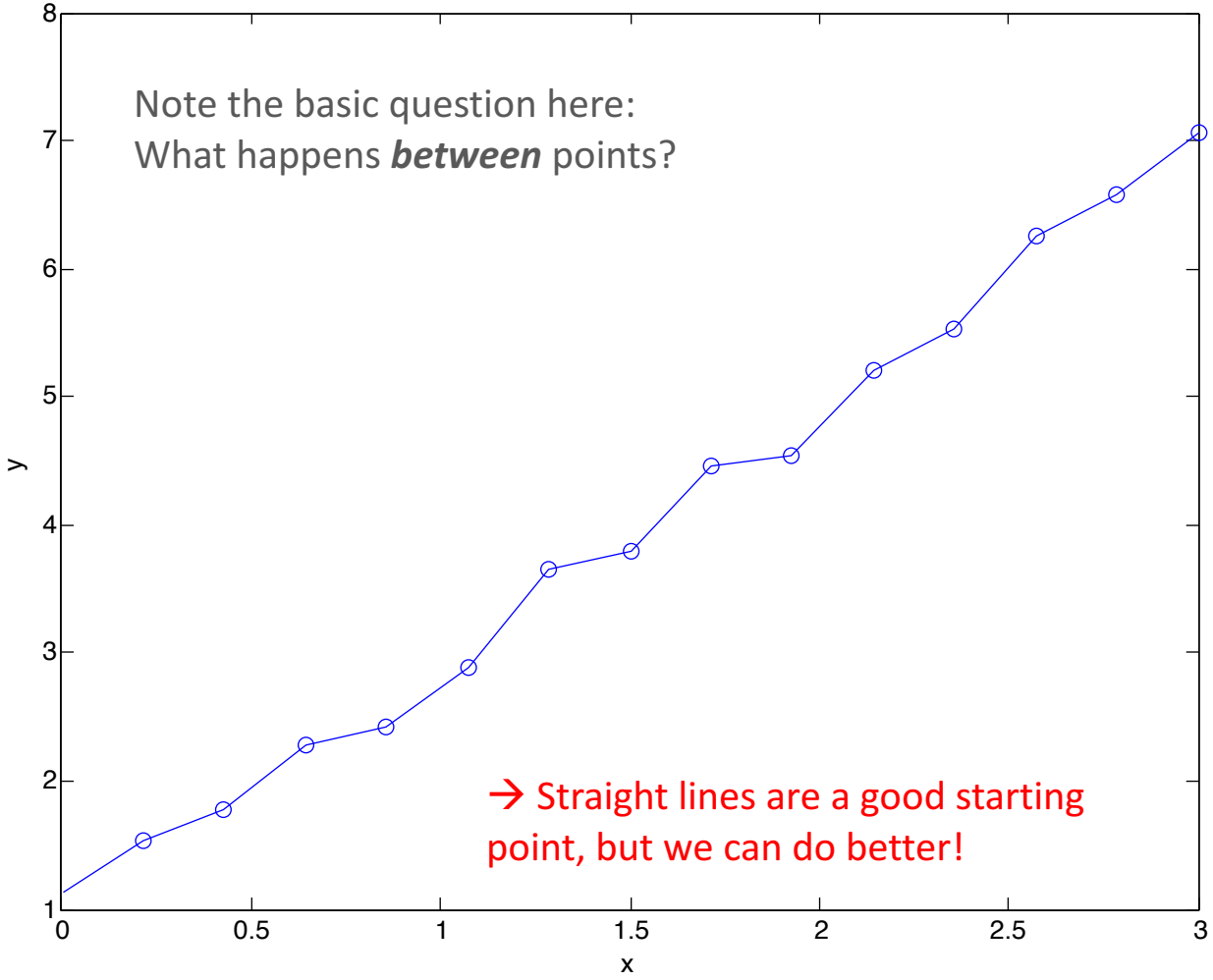
wikipedia (nonparametric regression)
Bergevin et al. (2010)

Aside: Interpolation

```
x=linspace(0,3,15);
y= 1+2*x+ 0.1*randn(numel(x),1)';
plot(x,y,'o-')
```

Interpolation –
The process of determining the value of a function between two points at which it has prescribed values

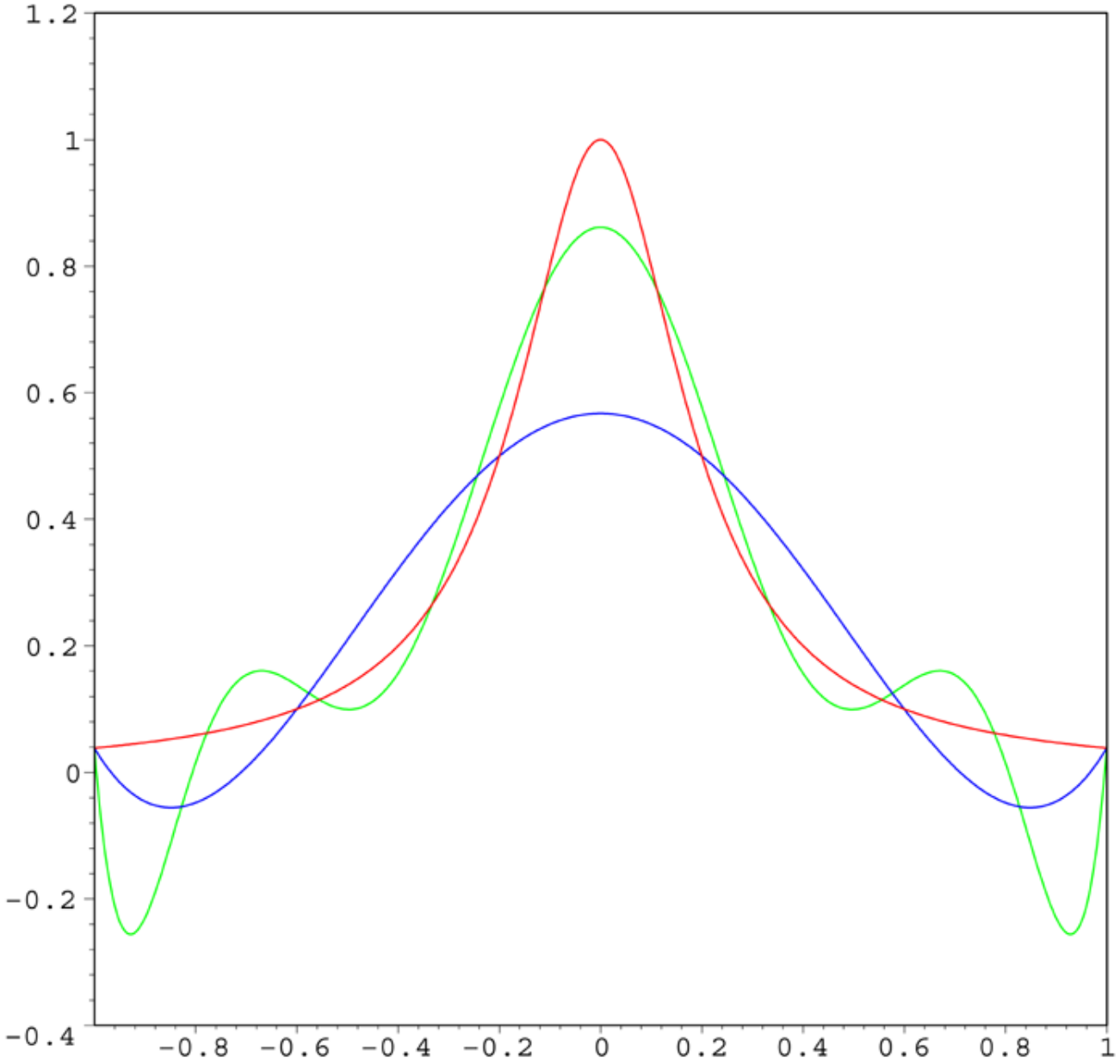
- What is the line between points?
- What exactly does plot.m do?
- Can we determine a 'better' curve?



‘Polynomial wiggle’

- Problem: Fitting with (high-ish order) polynomials can be problematic

The red curve is the Runge function. The blue curve is a 5th-order interpolating polynomial (using six equally spaced interpolating points). The green curve is a 9th-order interpolating polynomial (using ten equally spaced interpolating points). At the interpolating points, the error between the function and the interpolating polynomial is (by definition) zero. Between the interpolating points (especially in the region close to the endpoints 1 and −1), the error between the function and the interpolating polynomial gets worse for higher-order polynomials.

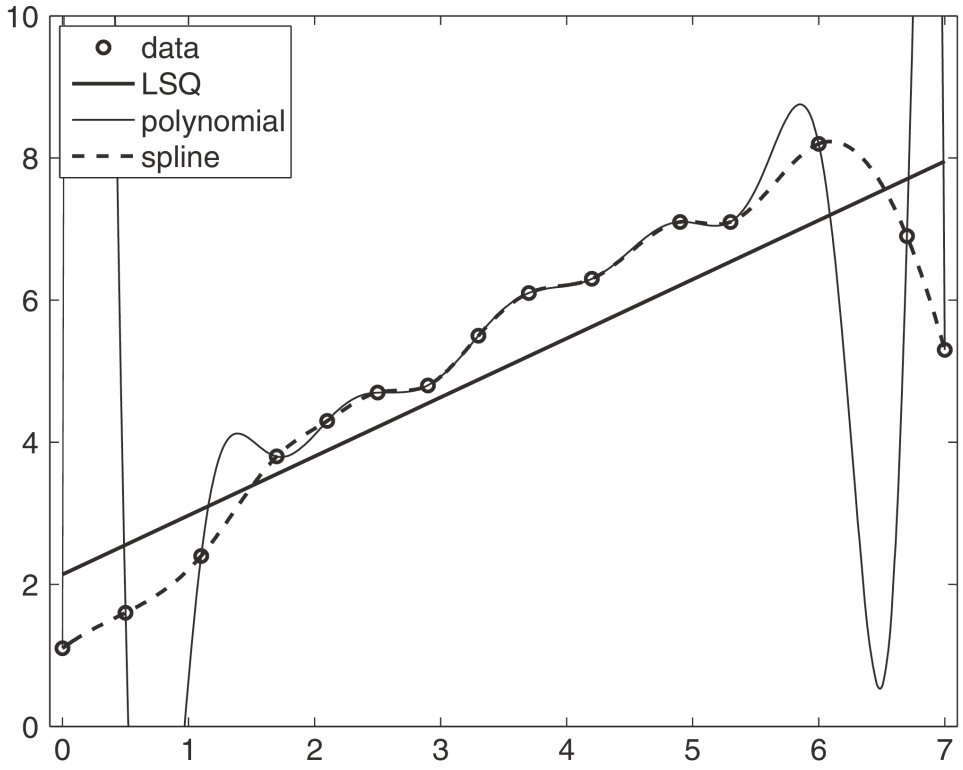


wikipedia (Runge's phenomenon)

Splines

- A form of ‘smoothing’ to interpolate noisy data

- “A **spline** or the more modern term **flexible curve** consists of a long strip fixed in position at a number of points that relaxes to form and hold a smooth curve passing through those points for the purpose of transferring that curve to another material.” [wikipedia (flat spline)]



- Basic idea is to do a ‘local’ fit, using a polynomial of low-enough order to avoid ‘wiggle’, but high enough to capture the basic features (i.e., curvature) of the data

→ Think carefully about the three different types of curves above

Kutz (2013)

Splines

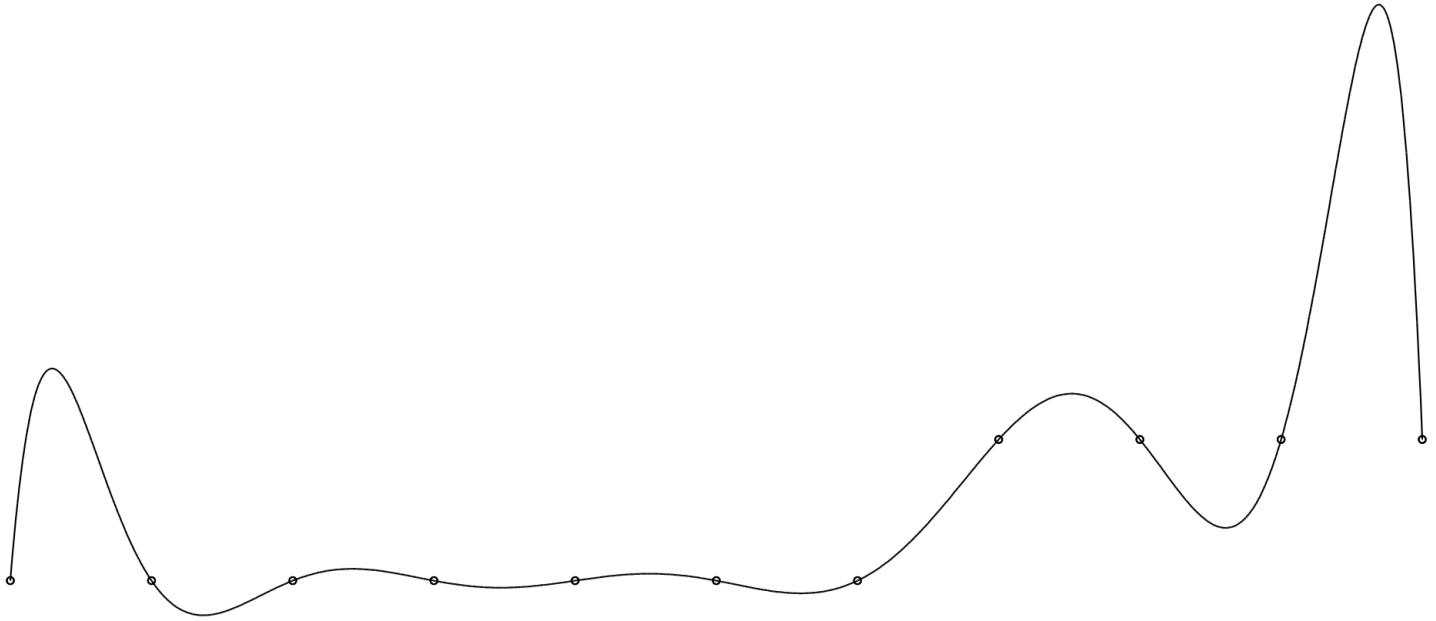


Figure 1: Lagrange interpolation of data points

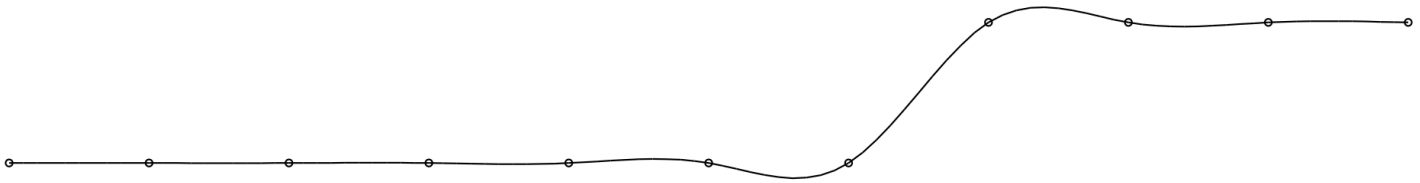


Figure 2: Spline interpolation of the same data points

http://www.math.ucla.edu/~baker/149.1.02w/handouts/dd_splines.pdf

Cubic splines

- By default, Matlab's `spline.m` uses a cubic polynomial

Basic idea: Cubic provides good balance between simplicity and capturing curvature

"A tridiagonal linear system (with, possibly, several right sides) is being solved for the information needed to describe the coefficients of the various cubic polynomials which make up the interpolating spline. spline uses the functions ppval, mkpp, and unmkpp. These routines form a small suite of functions for working with piecewise polynomials."

- Basic algorithm consists of:

  - Fitting a cubic polynomial to a limited/centered group of data points
  - Set various properties (e.g., 2nd derivative) equal at the endpoints to guarantee smoothness ('piecewise continuity')
  - Solve the resulting (linear) equations (which typically have a nice matrix form)

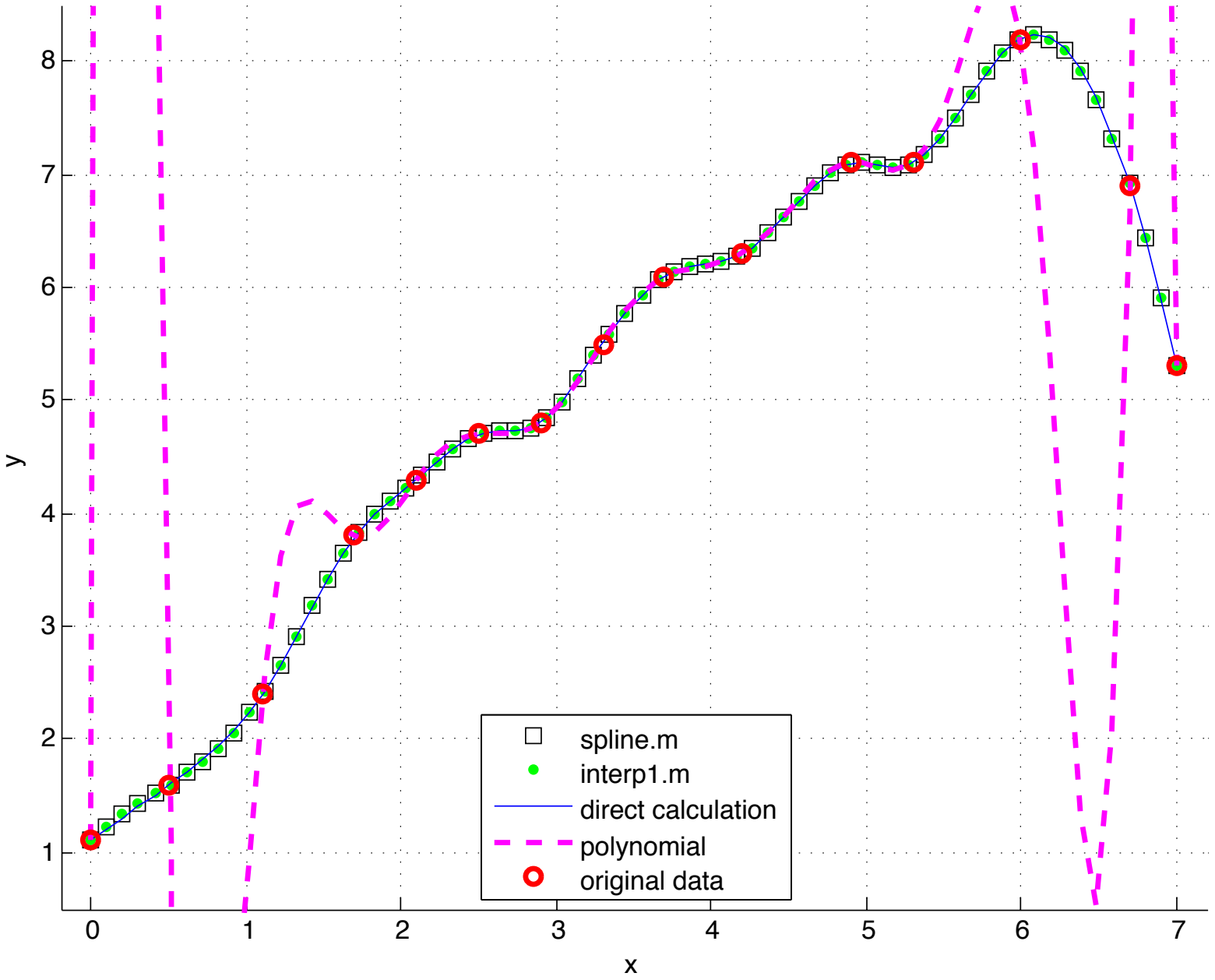
Note: Curve fitting is so important/ubiquitous, Matlab has a whole separate 'Curve Fitting Toolbox'

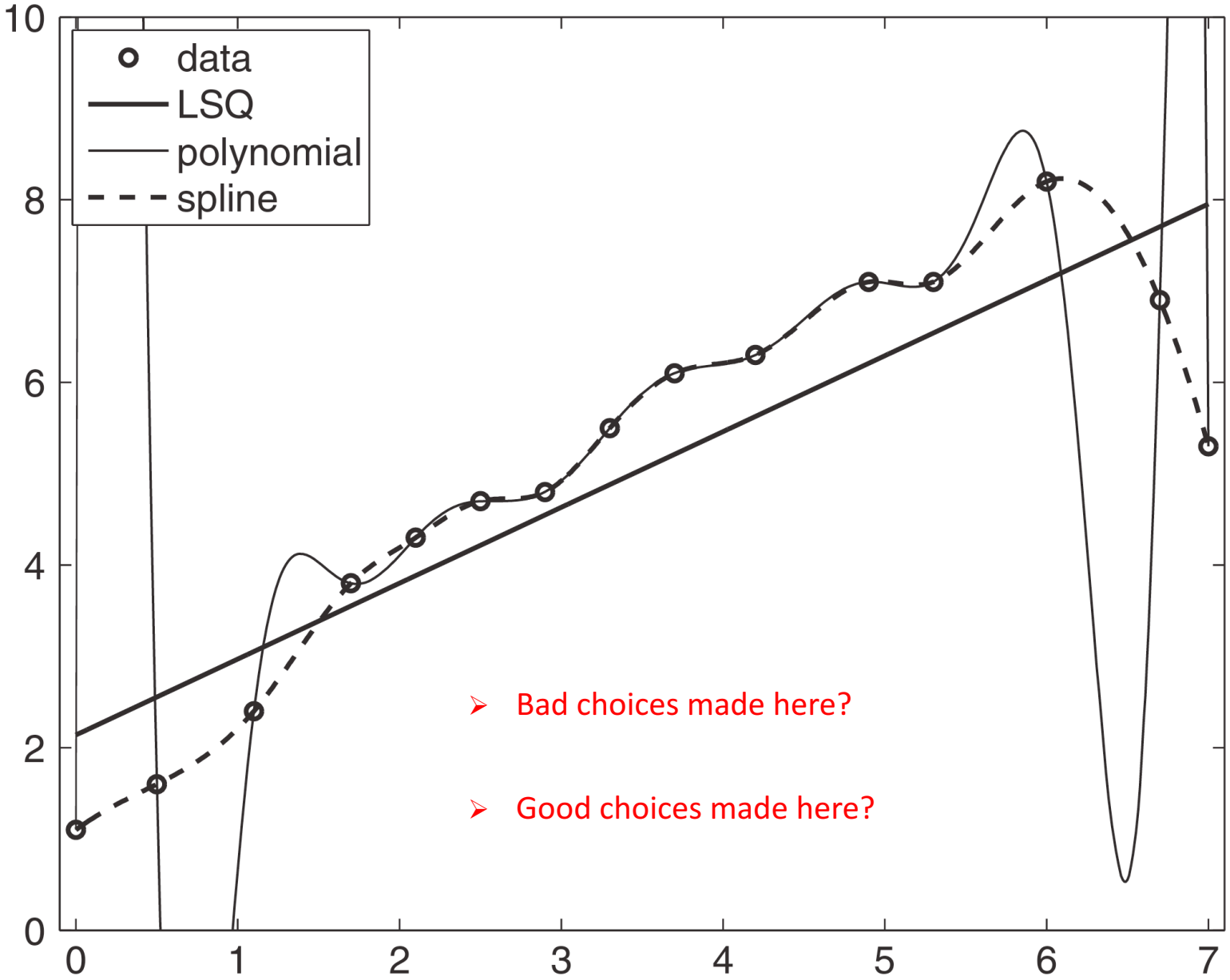
http://www.mathworks.com/help/matlab/ref/spline.html

```
% ### EXsplines1.m ###     10.09.14
% Determine cublic spline fit to a set of (specified) data. Does this three
% ways:
% Method 0 - via built-in Matlab spline.m function
% Method 1 - via built-in Matlab interp1.m function
% Method 2 - directly coded
% Method 3 - polynomial of order n-1 (where n=numel(x))

% source code for doing cubic spline (sans Matlab spline.m) is
% http://m2matlabdb.ma.tum.de/cspline.m?MP_ID=14

clear; figure(1); clf;
% ----------------------------
% User Inputs

N= 70;     % # of points for fit (over interval [min(x) max(x)])

% define 'data' (see pg.69 from Kuts, 2013)
x= [0 0.5 1.1 1.7 2.1 2.5 2.9 3.3 3.7 4.2 4.9 5.3 6.0 6.7 7.0];
y= [1.1 1.6 2.4 3.8 4.3 4.7 4.8 5.5 6.1 6.3 7.1 7.1 8.2 6.9 5.3];

% ------------------------------------------------
xx= linspace(min(x),max(x),N); % determine fit x-values

% ------------
% Method 0: Built-in Matlab function spline.m
yspline0= spline(x,y,xx);

% ------------
% Method 1: Built-in Matlab function interp1.m
 yspline1= interp1(x,y,xx,'spline');

% Note: Other options for interp1 exists, such as:
% > yspline1= interp1(x,y,xx);  % straight line nearest neighbor interp (default for plot.m)
% > yspline1= interp1(x,y,xx,'nearest');  %
```

```
% ------------
% Method 2: Direct computation of splines (see comments at top for source)
n = numel(x);   % # of points to be fit
if n ~= length(y)-2 & n ~= length(y)
  error(['y has to be of length length(x) + 2 or length(x)']);
end
if n < 2, error('only one value given, can not interpolate'); end
% check for the slopes at the endpoints being given or not
[nr,nc] = size(y);
if nr == 1,  y = reshape(y,nc,1); nr= nc; end
[nr,nc] = size(x);
if nr == 1,  x = reshape(x,nc,1); nr= nc; end
if(length(y) == length(x))
  naturalInterpolation = 1;
  dy_l = 0;
  dy_r = 0;
else
  naturalInterpolation = 0;
  % y consists of the slopes at the endpoints and of the values of y
  dy_l = y(1);
  dy_r = y(n+2);
  y = y(2:n+1);
end
if size(x) ~= size(y), error('x and y are of different size'); end
dx = [0; diff(x); 0];
dxx = dx(1:n) + dx(2:n+1);
% assemble matrix and rhs
M = spdiags([[dx(2:n)./dxx(2:n); 0] 2*ones(n,1) [0; dx(2:n)./dxx(1:n-1)]], -1:1, n,n);
% compute the rhs using aitken-neville scheme
% c    : second derivative
% a = y: values of y
% b    : first derivative
% d    : third derivative
b = diff(y) ./ dx(2:n);
c = 6 * diff([dy_l; b; dy_r])./ dxx;
% For natural spline interpolation
if(naturalInterpolation == 1)
  c(1)     = 0;
  c(n)     = 0;
  M(1,2)   = 0;
  M(n,n-1) = 0;
end
c = M\c;
d = diff(c)./dx(2:n);
b = b  - dx(2:n).* (c(1:n-1)/3 + c(2:n)/6);

% now compute the values yy (i.e., the fit)
yspline2 = zeros(size(xx));
for i=1:nr-1
  I = find(xx <= x(i+1) & xx >= x(i));
  yspline2(I) = y(i) + b(i)*(xx(I)-x(i))+c(i)/2*(xx(I)-x(i)).^2 + ...
          d(i)/6*(xx(I)-x(i)).^3;
end
```

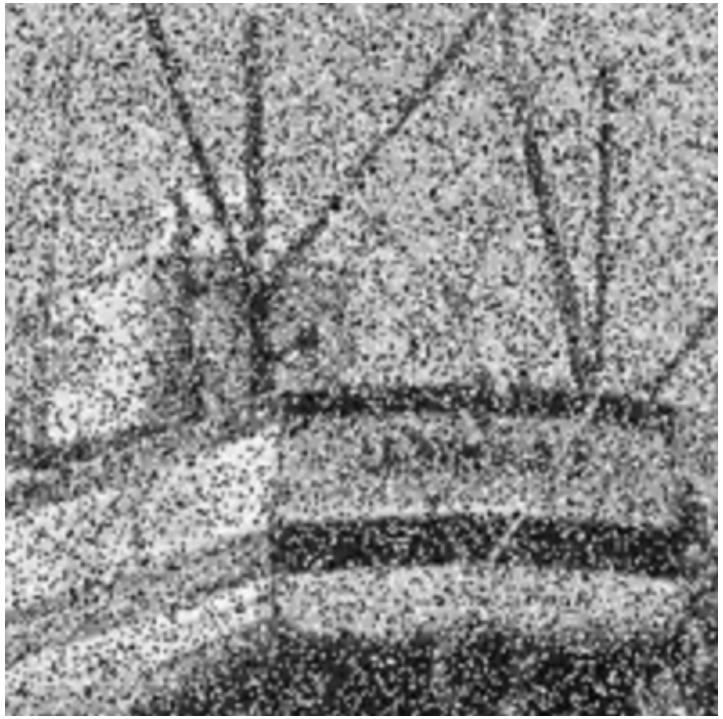
```
% ------------
% Method 3: polynomial fit via built-in Matlab function polyfit.m
coeff= polyfit(x,y,numel(x)-1);
ypoly= polyval(coeff,xx);

% ------------
% plot the data
figure(1); clf
hold on; grid on;
xlabel('x');ylabel('y');
plot(xx,yspline0,'ks','MarkerSize',6)
plot(xx,yspline1,'g.')
plot(xx,yspline2);
plot(xx,ypoly,'m--','LineWidth',2)
plot(x,y,'ro','MarkerSize',7,'LineWidth',2);
axis([-0.2 7.2 0.5 8.5])
```

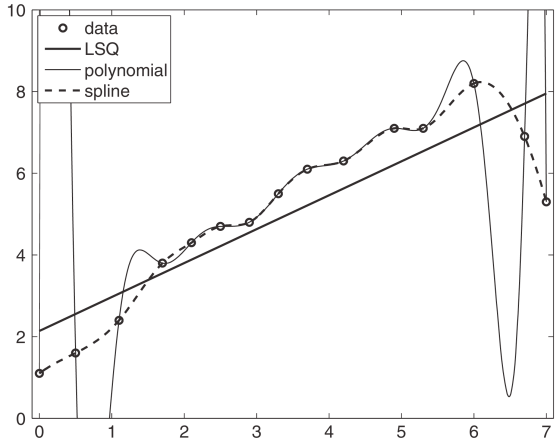

spline.m
interp1.m
direct calculation
polynomial
original data
x
y
0
1
2
3
4
5
6
7
1
2
3
4
5
6
7
8


data
LSQ
polynomial
spline
10
8
6
4
2
0
0
1
2
3
4
5
6
7
Bad choices made here?
Good choices made here?

Kutz (2013)

Aside: De-noising images





- De-noising (or "smoothing") can arise in a variety of contexts….

Locally-weighted polynomial regression (loess)

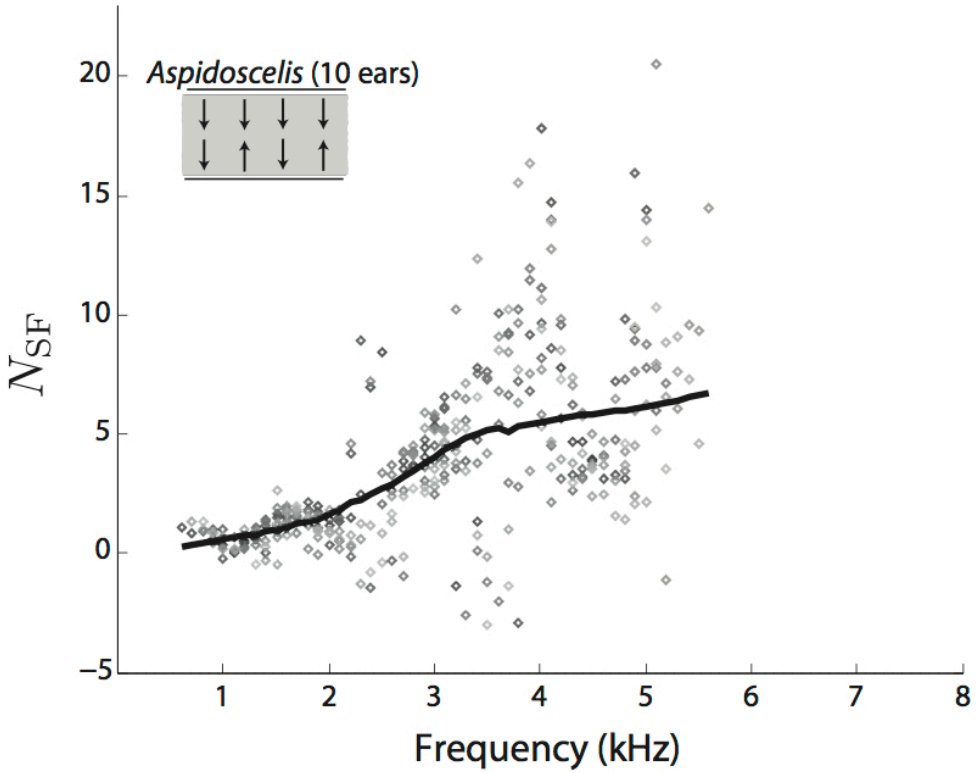
- Basic idea: Do a 'local' regression (loess = '**LO**cal regr**ESS**ion') to get a smooth trend curve

  → chiefly dictated by the data, not any 'model' assumptions

- Perform a least-squares fit to ***localized subsets*** of the data. Can also allow for weighting to be factored in (just like the standard deviation we saw for linear regression)

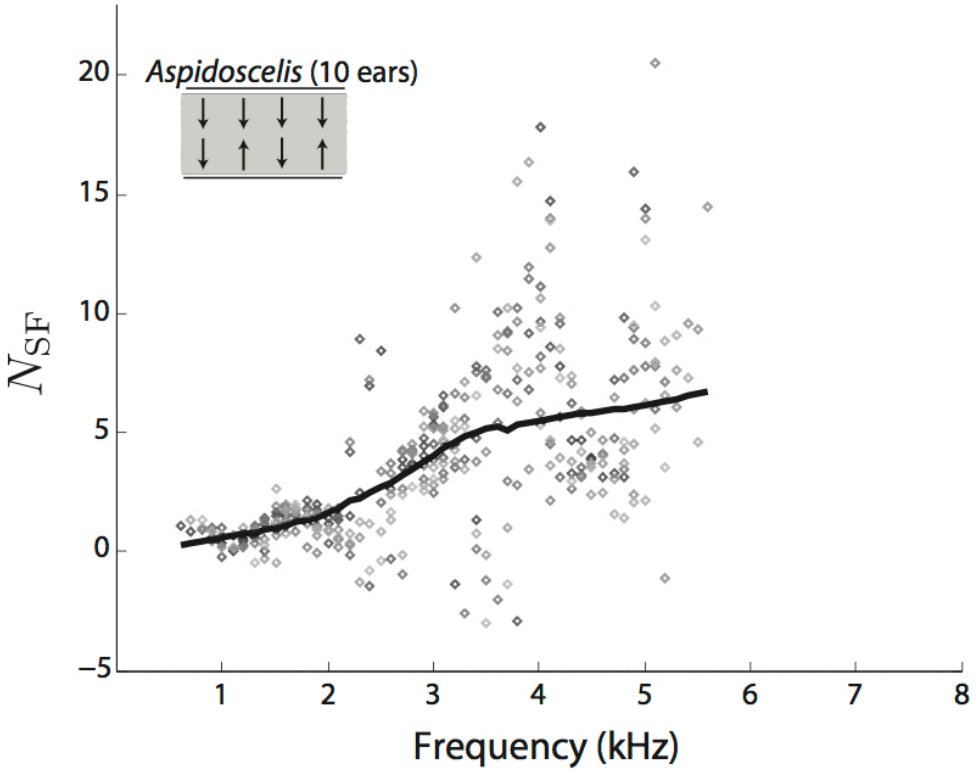
- Similar in spirit to splines, just a bit more general

→ [Reemphasis] This sort of situation is very common in numerous scientific and engineering applications!



Locally-weighted regression (loess)

- Has both:
- ***Pros*** (e.g., model-independent, determines interpolative trend from data directly) &
- ***Cons*** (e.g., does not 'model' the data per se, can be computationally intensive/inefficient)



- Can also be sensitive to outliers, so be careful (be smart up front about how to handle your data!)

'Golden rules' #1 – **The computer only does what you tell it to do**

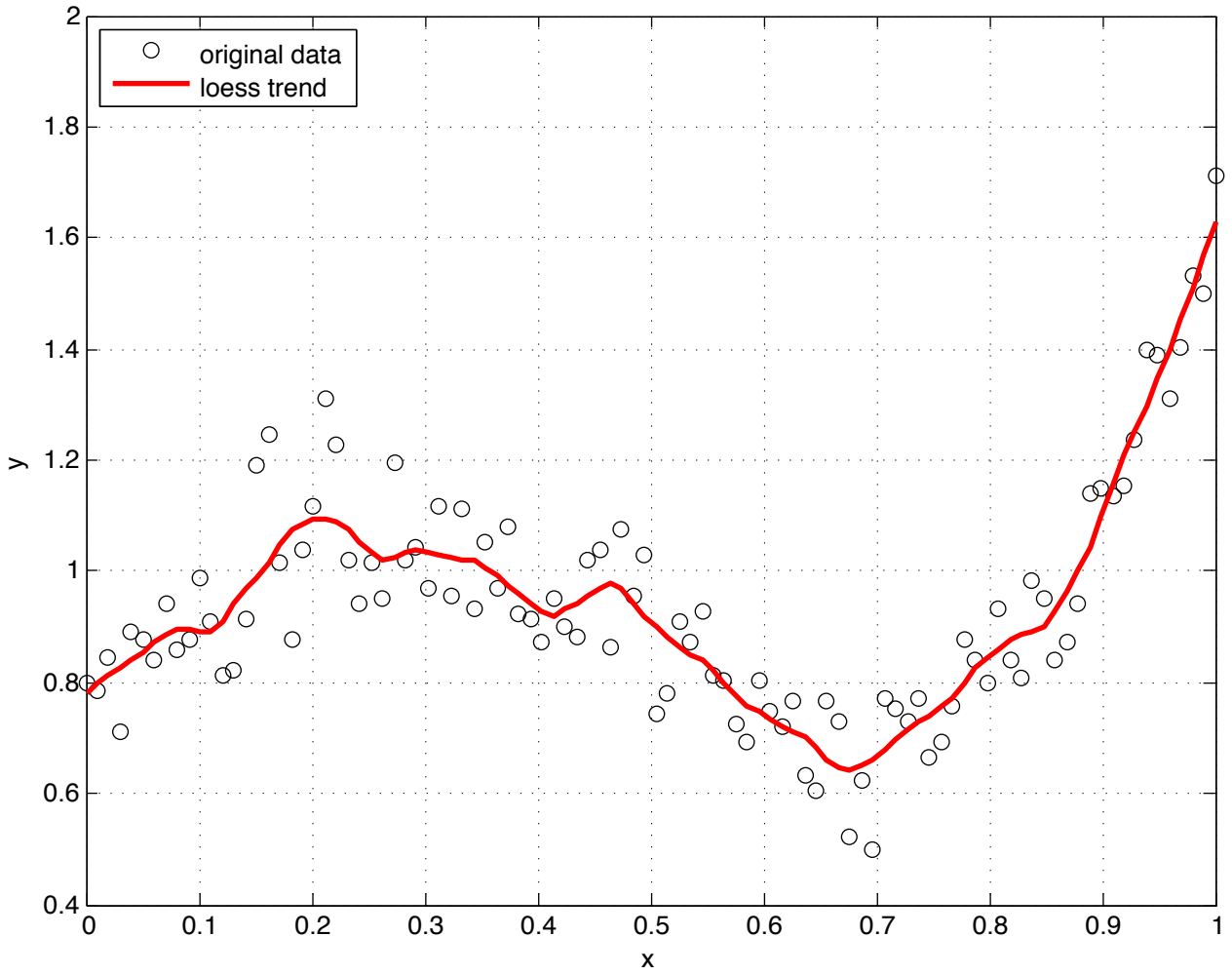
```
% ### EXloess1.m ###     10.12.14
% Determine loess fit to a set of randomly-generated noisy data

% Note: Requires loess.m [hacked together by C. Shera]

clear; figure(1); clf;
% ----------------------------
% User Inputs

N= 100;     % # of points for fit (over interval [min(x) max(x)])
xR= [0 1];  % range of x-values
scaleN= 0.1;    % scale factor for noise

alpha= 0.1;     % loess fit parameter (between 0 and 1)
order= 1;        % order of polynomial for fit

% ------------------------------------------------
x= linspace(min(xR),max(xR),N); % determine fit x-values
polyS= ceil(10*rand(1));   % randomly determine polynomial order
Pv= 2*rand(polyS,1)-1;  % polynomial coefficients
y= polyval(Pv,x)+ rand(1)*sin(2*pi*x)+ scaleN*randn(N,1)'; % pseudo-random 'data'

% ---
xFit= linspace(min(xR),max(xR),N); % can 'resample' if desired (i.e., need not have xFit=x)
yFit= loess(x,y,xFit,alpha,order,[],1,0,0.1);   % loess fit via external function

% ---
% visualize
plot(x,y,'ko'); hold on; grid on;
plot(xFit,yFit,'r-');
xlabel('x'); ylabel('y');
legend('original data','loess trend');
```

- What type of polynomial is being used here?

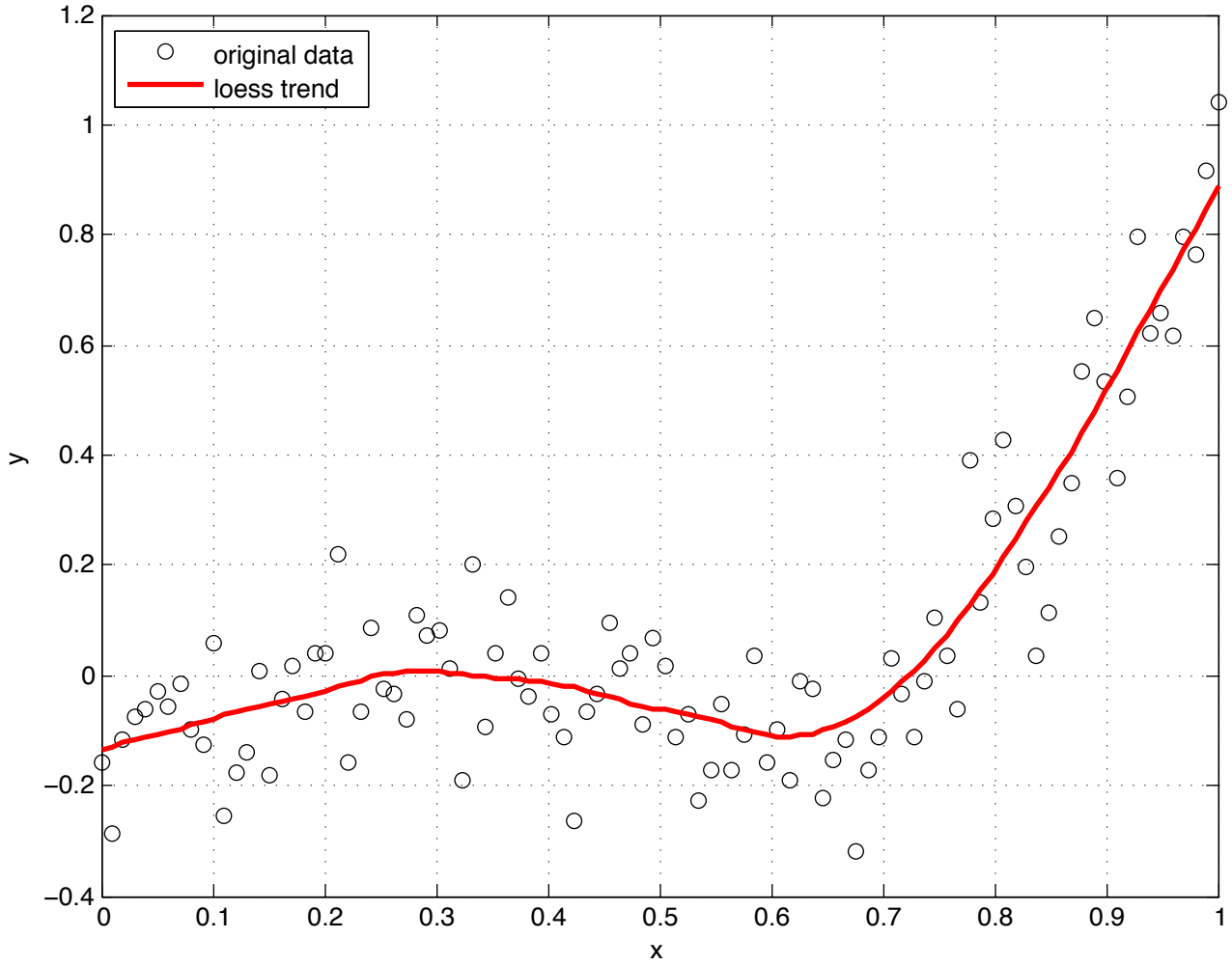
```
function y = loess (xi,yi,x,alpha,lambda,weights,robust,collapse,dither)
% y = loess (xi, yi, ?x=unique(xi)?, ?alpha=0.1?, ?lambda=1?, ?weights=[]?,
%            ?robust=0?, ?collapse=0?, ?dither=0?)
% Returns the loess fit y(x) to data points yi(xi) having optional
% weights (e.g., weights = 1/sigma_yi^2). The smoothing
% parameter alpha (alpha<1) is the fraction of the total number
% of data points to fit. Typically, alpha<<1 (so that the
% fit is *local*). Uses a polynomial fit of order lambda.
% Note that the data (xi,yi) do NOT need to be pre-sorted by xi,
% (but its a good idea to do so if you intend to plot the results!)
%
% The robust option computes an iterated robust loess fit.
% First, an initial (un-robust) loess fit is performed. The
% weights are then modified (based on the residuals to reduce
% the influence of outliers) and a new fit performed.
% Ideally, the cycle of fitting and adjusting weights based on
% residuals is iterated until the fit converges.
% Here, the fit is iterated a total of robust times;
% in practice robust=1 often suffices. If there are lots of
% data points, a robust fit can take a long time.
%
% The collapse option collapses all data points at the same
% value of xi and replaces the multiple yi by their median value.
% (We use the median not the mean in the hope that it'll be
% less sensitive to outliers.) Collapsing helps maintain
% the 'local' character of the fit in cases where many xi have
% multiple corresponding yi.
% (Note that the computation of the mean or median should take the weights
% into account and new weights should be computed using the variance.
% But we need to figure out what weights should be assigned to non-repeated
% values.)
%
% The dither option dithers the xi randomly by dither percent
% using a Gaussian distribution. Dithering is an alternative
% solution to the 'duplicate xi problem' addressed by collapsing.
% If dither is set, collapse is unset.
%
% Reference:
%  Visualizing Data, William S. Cleveland
%  Hobart Press, 1993, pg 100ff
%
% Hacked together by C.A. Shera
```

```
function y = loess (xi,yi,x,alpha,lambda,weights,robust,collapse,dither)
% y = loess (xi, yi, ?x=unique(xi)?, ?alpha=0.1?, ?lambda=1?, ?weights=[]?,
%            ?robust=0?, ?collapse=0?, ?dither=0?)

%% Hacked together by C.A. Shera
%

  if (nargin < 3 | isempty (x)),       x = unique(xi);           end;
  if (nargin < 4 | isempty (alpha)),   alpha = 0.1;              end;
  if (nargin < 5 | isempty (lambda)),  lambda = 1;               end;
  if (nargin < 6 | isempty (weights)), weights = ones(size(yi)); end;
  if (nargin < 7 | isempty (robust)),  robust = 0;               end;
  if (nargin < 8 | isempty (collapse)),collapse = 0;             end;
  if (nargin < 9 | isempty (dither)),  dither = 0;               end;

  if (dither), collapse=0; end

  % collapse repeated values...
  if (collapse)
    [sxi,I,J] = unique (xi);
    syi = zeros(size(sxi));
    wgt = zeros(size(sxi));
    for k = 1:length(sxi)
      avg_me = find (J==k);
      syi(k) = median (yi(avg_me)); % should use weights
      wgt(k) = mean (weights(avg_me));  % coarse
    end
    xi = sxi;
    yi = syi;
    weights = wgt;
  end

  % dither the xi...
  if (dither ~= 0)
    xi = xi .* (1 + dither*randn(size(xi))/100);
  end

  n = length (xi);
  q = min (n, round (abs (alpha)*n));

  % do sanity control on the robust flag since
  % we later use it to control the iteration...
  robust = round (abs (robust));

  weights = abs (weights);      % must be non-negative
  original_weights = weights;       % save for later

  if (~robust)
    % allocate some space for the fit...
    y = zeros (size (x));
  end

  % we dither the xi by a small random amount to prevent
  % there from being lots of repeated xi values, which can cause
  % havoc when trying to determine the q closest values...
  %  xi = xi .* (1 + 1e-6*randn(size(xi)));

  % iterate to obtain robust fits at the points xi...
  while (robust)
    robust = robust - 1;

    % do a normal loess fit at points xi (not x) using the current weights
    y = loess (xi,yi,xi,alpha,lambda,weights,0,collapse);

    % compute residuals from fit...
    res = yi - y;

    % Compute bisquare robustness weighting function using the residuals...
    % Outliers (points with large residuals) receive a weighting near zero.
    mar = median (abs (res));       % median absolute residual
    u = res / (6*mar);

    % before iterating the fit, modify the original weights
    % using the bisquare robustness weighting...
    weights = original_weights .* bisquare (u);
  end

  y = zeros (size(x));          % allocation

  % after this loop, robust is always zero...
  % Loop over x (not xi) ...
  flag=0;
    for i = 1:length (x)
      % Compute the tricube weighting functions...
      % Points are weighted by their distance from x(i) using a
      % variable window defined so that the qth most distant point
      % has w=0. Except near the ends, the window will typically
      % be roughly symmetric about x(i). Note that when determining
      % the qth most distant point, we count points at the same xi
      % as one. This differs from Cleveland's description, but is
      % more robust (and sensible?) in certain pathological cases.

      Delta = abs (xi-x(i));
      Delta_q = unique (Delta);       % sort array, removing duplicates

      % multiply in the tricube weighting...
      w = weights .* tricube (Delta/Delta_q(q));


      % Locate the points to fit...
      fit_me = find (w>0);

      % take care of various pathological cases...
      switch (length(fit_me))
       case 0
        if flag== 0 warning ('loess: Empty window! Using NaN.'); end
        y(i) = NaN;
        flag= 1;
        continue;

       case 1
        if flag== 0 warning ('loess: Single point in window! Skipping fit.'); end
        y(i) = yi(fit_me);
        flag= 1;
        continue;

       otherwise
        lam = min(lambda,length(fit_me)-1);
        if (lam ~= lambda)
          if flag== 0 warning ('loess: Too few points in window! Reducing lambda.'); end
          flag= 1;
        end

        % Do the fit...
        if (lam==1)
      % MUCH faster than polyfitw in linear case
      [a,b] = linear_fit (xi(fit_me),yi(fit_me),1./sqrt(w(fit_me)));
      y(i) = a + b*x(i);
        else
      p = polyfitw (xi(fit_me),yi(fit_me),lam,w(fit_me));
      y(i) = polyval (p,x(i));
        end
      end
    end
    return

% ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

function T = tricube (u)
  u = abs (u);
  T = zeros (size (u));
  i = find (u<1);
  T(i) = (1-u(i).^3).^3;
  return

% ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

function T = bisquare (u)
  u = abs (u);
  T = zeros (size (u));
  i = find (u<1);
  T(i) = (1-u(i).^2).^2;
  return
```

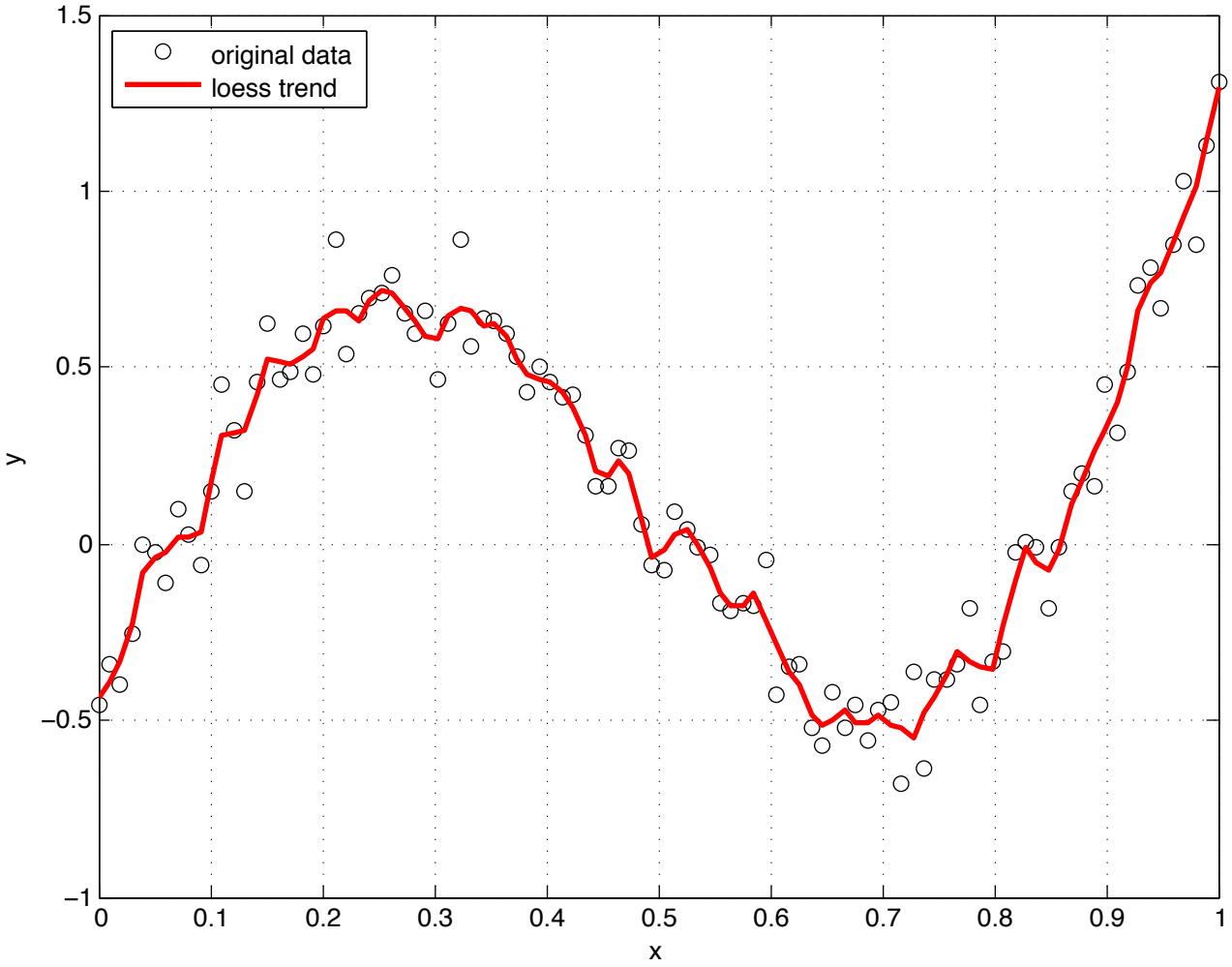
```
alpha= 0.1;          yFit= loess(x,y,xFit,alpha,1,[],1,0,0.1);
```



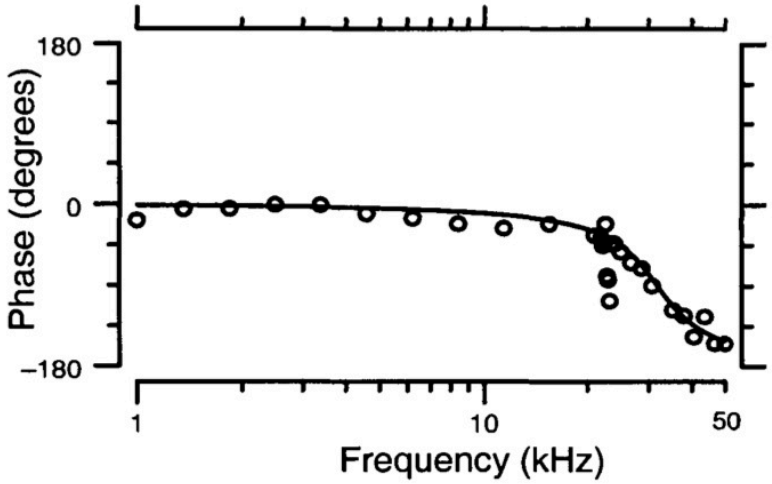
```
alpha= 0.3;
```


original data
loess trend
1.2
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x

```
alpha= 0.05;
```


original data
loess trend
1.5
1
0.5
0
−0.5
−1
y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x

Summary (regression)



Starting point is some data, which you may want to fit a specific function to or determine a trend

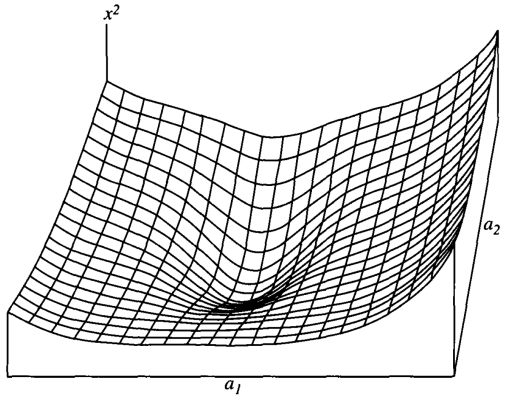
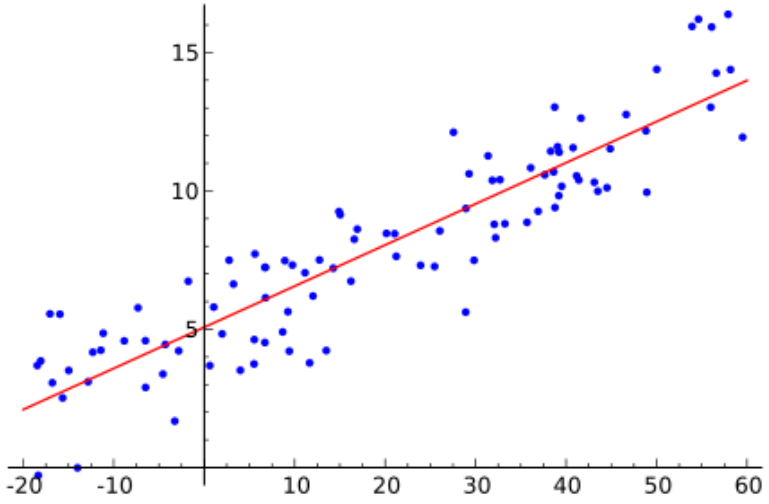
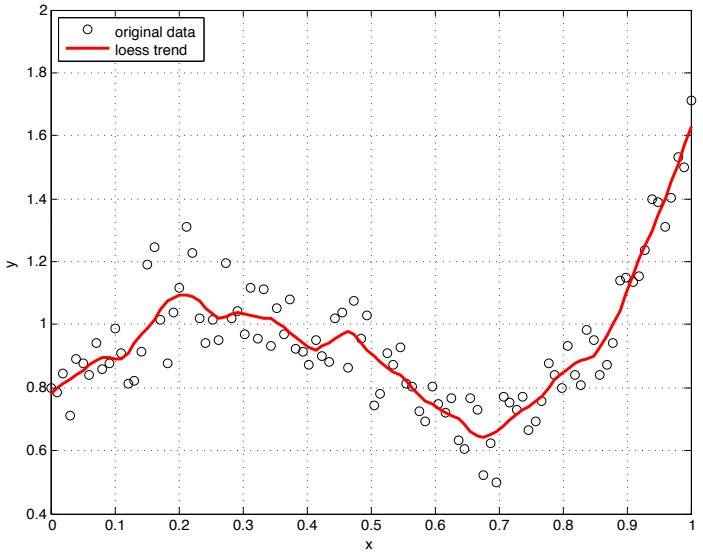


FIGURE 8.2
Chi-square hypersurface as a function of two parameters.

Regression can be linear or nonlinear, the latter leading to some tricky computational approaches



Key quantity in 'least squares' analysis is χ^2 ("chi-squared"), which you want to minimize



Non-parametric regression (e.g., cubic splines, loess) can be very useful for finding trends sans a 'model'

Post-class exercises

- What is the distinction (if any) between ‘interpolation’ and ‘curve fitting’?

- How might you modify EXregression6.m (or myTestFunction2.m) such that the fit to the micro-mechanical resonator steady-state phase is improved?

- Modify the loess code to see how the trend varies for different order polynomials. And what happens if you modify the other parameters (e.g., ‘robust’)?

- Modify the loess code to introduce a randomized uncertainty for each point and use such for a ‘weighting’ in computation of the trend

Coming up next....

EXfractal1.m