Least-Squares Fitting

Helwig Hauser *et al.*,
UiB Dept. of Informatics

Looking Back & Forth

Last time:

- variance and covariance
- eigenanalysis
- PCA (principal component analysis)

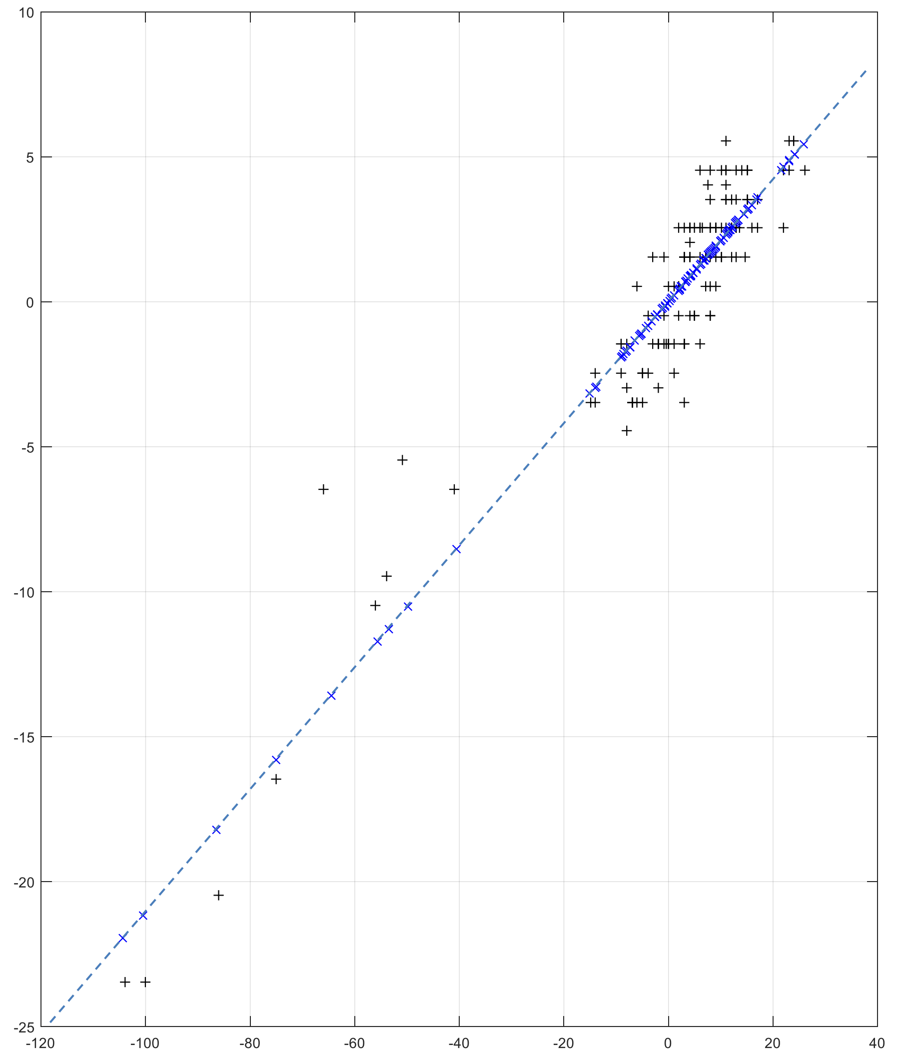
Today:

- least-squares fitting
  - best linear fit
  - best quadratic fit
  - normal equations
  - numerical issues

Introduction

Least-Squares Fitting:

- reducing data complexity:
  - find the line that approximates the data best
  - find some other (simple) function that approximates the data best
- modeling a phenomenon
- "solving" overdetermined equation systems



[example related to the lecture on PCA]

Systems of Linear Equations Revisited

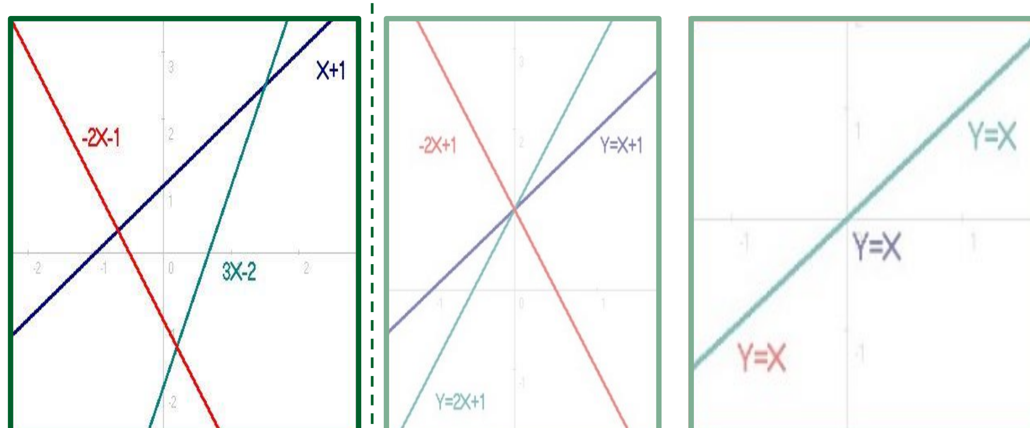
Given a system of linear equations $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ ($\mathbf{A} \in \mathbb{R}^{n\mathsf{x}m}$),

– we have discussed $n = m$ (one sol., no sol., or many solutions)

– we touched upon $n \neq m$

- if $n < m$ (underdetermined system):
  - – usually many solutions
  - – also possible: no solution
- if $n > m$ (overdetermined system):
  - – usually no solution
  - – also possible: one solution or even many

[illustration from Wikipedia]



Modeling the Data

Given some data, measuring $y(x)$ **relations**:

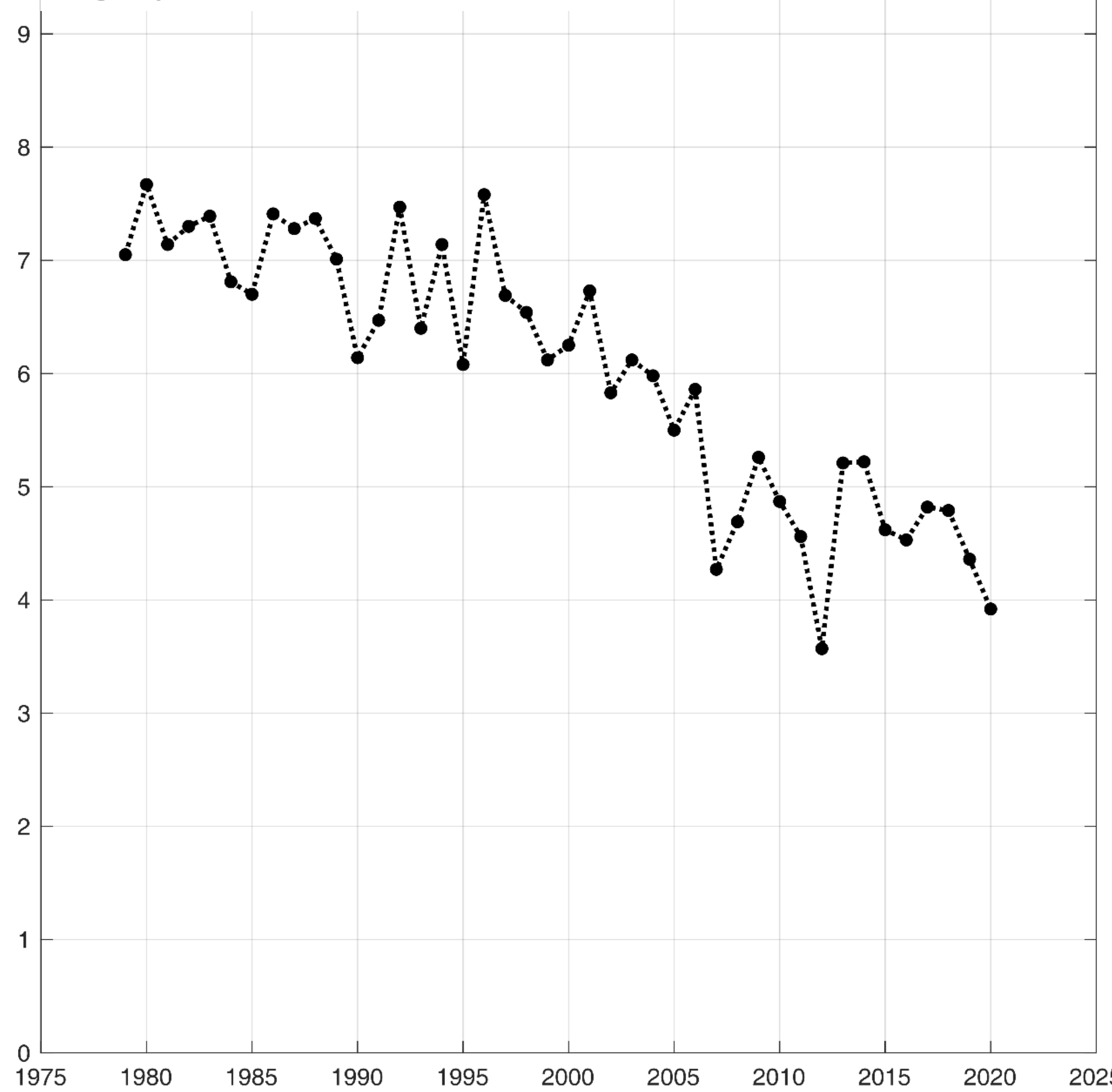
- here:
  time (years) on x
  vs. Sept. sea ice
  extent in the Arctic (y)
  [in 10^6 km^2]

Any trend?

- best-fit line?

Model:

- data x_i, y_i
- model $m(x) = kx + d$
- $y_i = m(x_i) + e_i$
- error e_i



Measuring the Error (deviation from model)

Key to finding "the optimal model", for ex.:

- maximum error $$E_\infty(f) = \max_{1<k<n} |f(x_k) - y_k|$$
- average error $$E_1(f) = \frac{1}{n}\sum_{k=1}^{n} |f(x_k) - y_k|$$
- RMS (root-mean-square) error $$E_2(f) = \left(\frac{1}{n}\sum_{k=1}^{n} |f(x_k) - y_k|^2\right)^{1/2}$$

Above:

- data (n samples): x_k and y_k
- model: $f(x)$ $\rightarrow$ $f(x_k)$ modelled to be similar to y_k

Measuring the Error/Deviation

Key to fin

– maxim

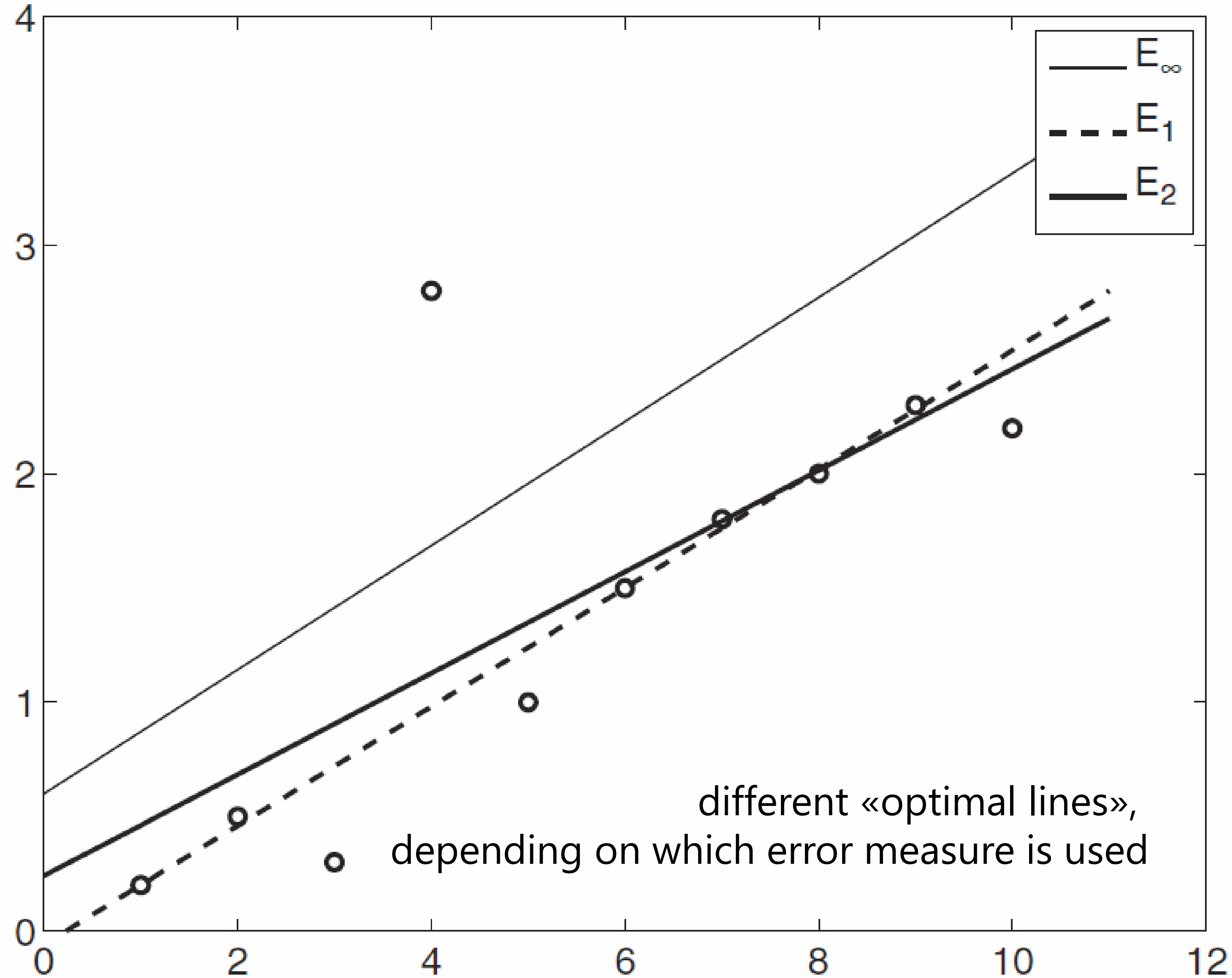
– averag

– RMS (r

Above:

– model:

– data: x



Least-squares Fit

Assuming E_2, i.e., RMS errors:

- we should minimize for all data $$E_2(f) = \left(\frac{1}{n} \sum_{k=1}^{n} |f(x_k) - y_k|^2 \right)^{1/2}$$

Minimizing squared errors → minimizing E_2!

- instead of minimizing $E_2(f)$ as above, we min. $\sum_{k=1}^{n} |f(x_k) - y_k|^2$

Best-fit line example:

- in order to fit a line model $f(x) = Ax + B$

  we minimize $$E_2(f) = \sum_{k=1}^{n} |f(x_k) - y_k|^2 = \sum_{k=1}^{n} (Ax_k + B - y_k)^2$$

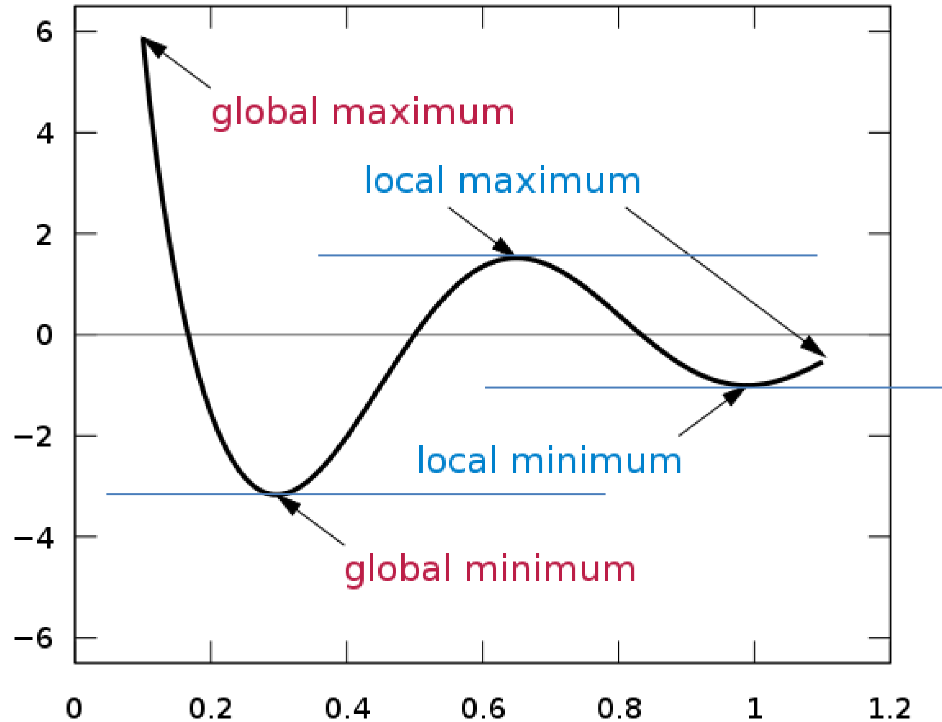
  (note the redefined, squared & scaled E_2)
- two unknowns here: A and B ← important to see the unknowns!
- the partial derivatives of E_2 wrt. A and B must be 0 in the min.!

Least-squares Fit

Assuming E_2, i.e., RMS errors:

– we should minimize $E_2(f) = \left(\frac{1}{n} \sum_{k=1}^{n} |f(x_k) - y_k|^2 \right)^{1/2}$ for all data

[illustration from Wikipedia]



Minimizing squared errors → n

– instead of minimizing $E_2(f)$ as

Best-fit line example:

– in order to fit a line model $f(x)$

we minimize $E_2(f) = \sum_{k=1}^{n} |f($

(note the redefined, squared E_2

– two unknowns here: A and B

– the partial derivatives of E_2 wrt. A and B must be 0 in the min.!

Inspecting the Partial Derivatives

Starting from the sum of squared errors E_2:

$$E_2(f) = \sum_{k=1}^{n} |f(x_k) - y_k|^2 = \sum_{k=1}^{n} (Ax_k + B - y_k)^2$$

- we derivate by both A and B and set to 0:

$$\frac{\partial E_2}{\partial A} = 0 : \quad \sum_{k=1}^{n} 2(Ax_k + B - y_k)x_k = 0$$

$$\frac{\partial E_2}{\partial B} = 0 : \quad \sum_{k=1}^{n} 2(Ax_k + B - y_k) = 0$$

- leading to a system of linear equations:

$$\underbrace{\begin{pmatrix} \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k \\ \sum_{k=1}^{n} x_k & n \end{pmatrix}}_{\text{values, computed from samples}} \begin{pmatrix} A \\ B \end{pmatrix} = \underbrace{\begin{pmatrix} \sum_{k=1}^{n} x_k y_k \\ \sum_{k=1}^{n} y_k \end{pmatrix}}_{\text{values, computed from samples}}$$

Sea Ice Example

42 years of measurements of Arctic sea ice extent:

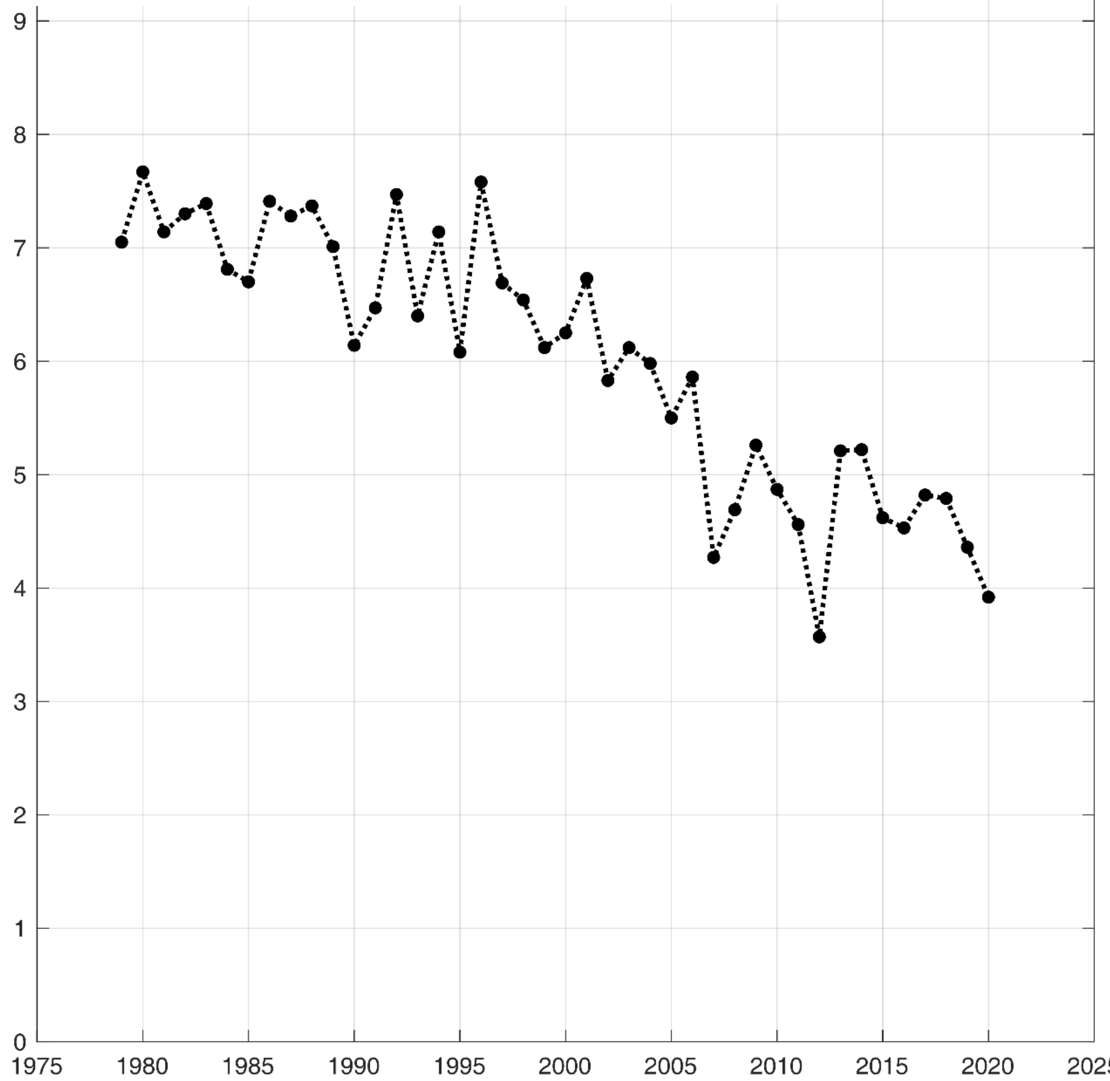
- $1979 \leq x_i \leq 2020$,
  $3.57 < y_i < 7.67$
- we set up the 2x2 equation system **M p** = **b**:

$$\mathbf{M} = \begin{pmatrix} \mathbf{x}^\top \mathbf{x} & \sum x_i \\ \sum x_i & n \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} \mathbf{x}^\top \mathbf{y} \\ \sum y_i \end{pmatrix}$$

- and solve then for **p**:

$$\mathbf{p} = \begin{pmatrix} -0.0837 \\ 173.3588 \end{pmatrix}$$



[see also MATLAB example 3a_A]

Sea Ice Example

42 years of measurements of Arctic sea ice extent:

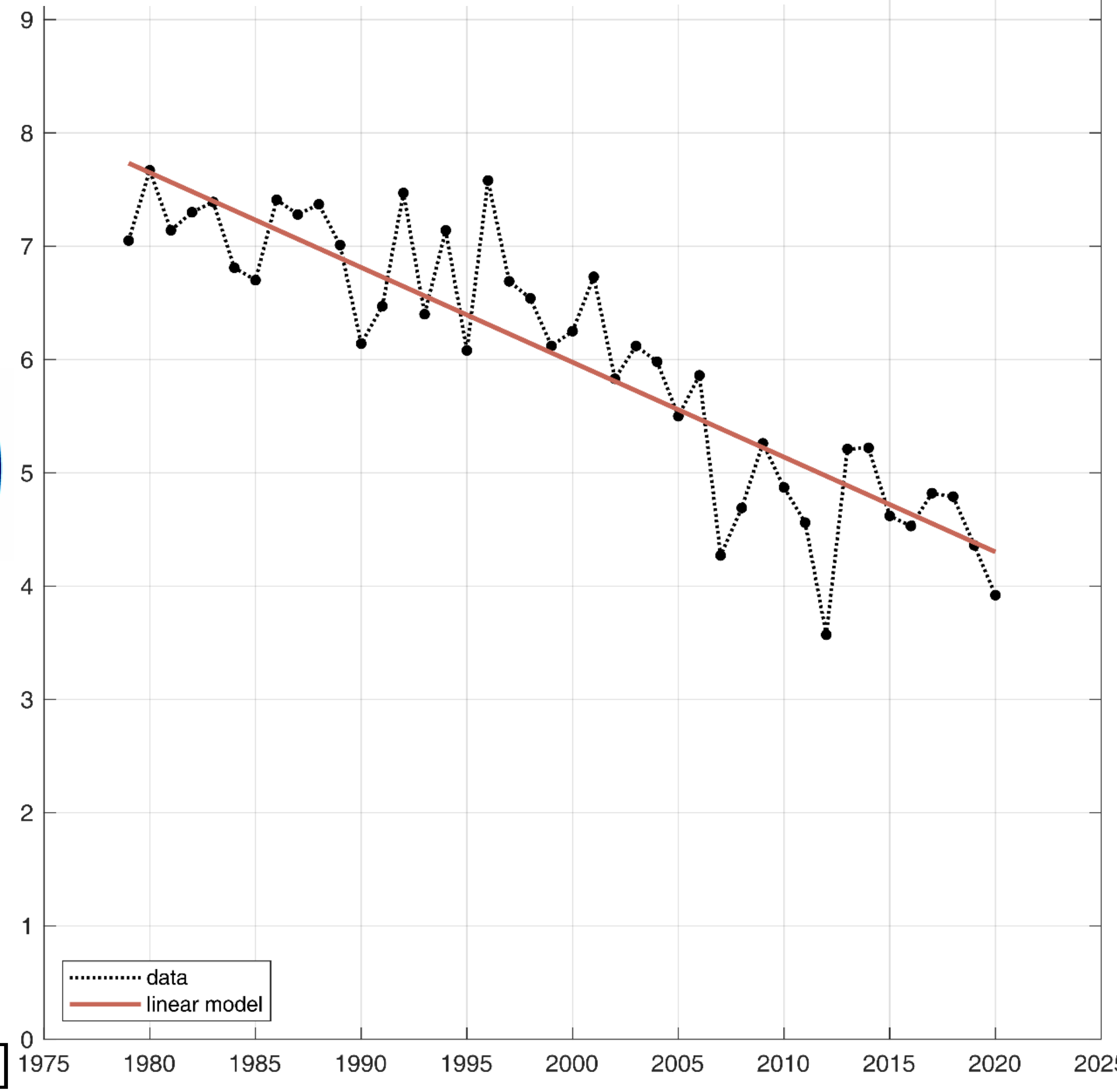
- $1979 \leq x_i \leq 2020$,
  $3.57 < y_i < 7.67$
- we set up the 2x2 equation system
  $\mathbf{M}\,\mathbf{p} = \mathbf{b}$:

$$\mathbf{M} = \begin{pmatrix} \mathbf{x}^\top \mathbf{x} & \sum x_i \\ \sum x_i & n \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} \mathbf{x}^\top \mathbf{y} \\ \sum y_i \end{pmatrix}$$

- and solve then for $\mathbf{p}$:

$$\mathbf{p} = \begin{pmatrix} -0.0837 \\ 173.3588 \end{pmatrix}$$



[see also MATLAB example 3a_A]

Sea Ice Example

42 years of measurements of Arctic sea ice extent:

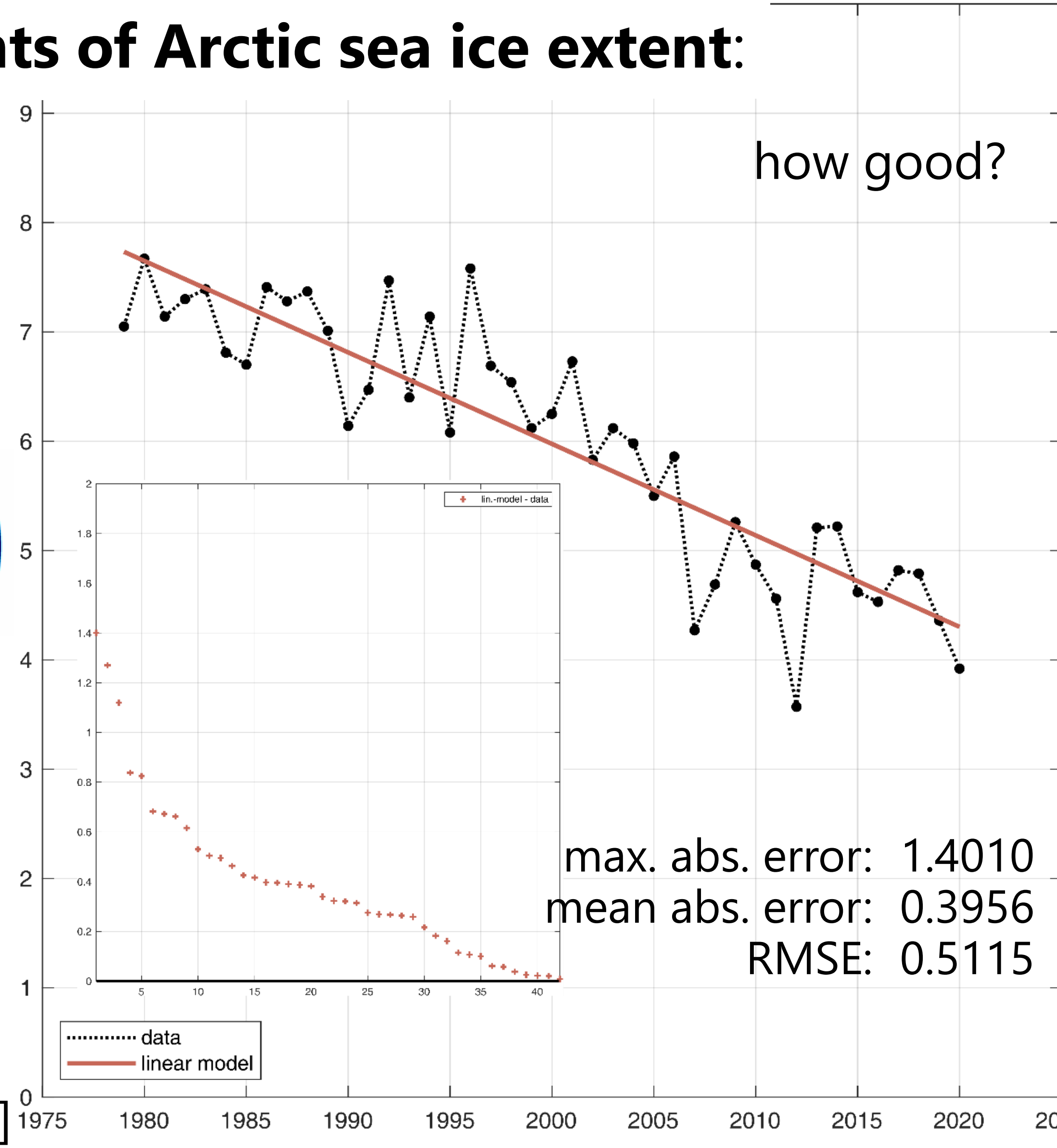
- $1979 \le x_i \le 2020$, $3.57 < y_i < 7.67$
- we set up the 2x2 equation system $\mathbf{M}\,\mathbf{p} = \mathbf{b}$:

$$\mathbf{M} = \begin{pmatrix} \mathbf{x}^\top\mathbf{x} & \sum x_i \\ \sum x_i & n \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} \mathbf{x}^\top\mathbf{y} \\ \sum y_i \end{pmatrix}$$

- and solve then for $\mathbf{p}$:

$$\mathbf{p} = \begin{pmatrix} -0.0837 \\ 173.3588 \end{pmatrix}$$



[see also MATLAB example 3a_A]

Other, non-linear Models

[see also MATLAB example 3a_B]

Alternative to linear regression:

– fitting a higher-order polynomial, for ex.: $f(x) = A\,x^2 + B\,x + C$

– now: three unknowns A, B, and C

– considering the partial derivatives of E_2 wrt. A, B, and C, and setting them to 0, gives a 3 x 3 system of equations

$$E_2(A,B,C) = \sum_k \left(f(x_k) - y_k\right)^2 = \sum_k \left(Ax_k^2 + Bx_k + C - y_k\right)^2$$

$$\frac{\partial E_2}{\partial A} = 0: \quad \sum\nolimits_k 2\left(Ax_k^2 + Bx_k + C - y_k\right)x_k^2$$

$$\frac{\partial E_2}{\partial B} = 0: \quad \sum\nolimits_k 2\left(Ax_k^2 + Bx_k + C - y_k\right)x_k$$

$$\frac{\partial E_2}{\partial C} = 0:$$

$$\boxed{\begin{pmatrix} \sum_k x_k^4 & \sum_k x_k^3 & \sum_k x_k^2 \\ \sum_k x_k^3 & \sum_k x_k^2 & \sum_k x_k \\ \sum_k x_k^2 & \sum_k x_k & n \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} \sum_k x_k^2 y_k \\ \sum_k x_k y_k \\ \sum_k y_k \end{pmatrix}}$$

Other, non-linear Models

Alternative to linear regression:

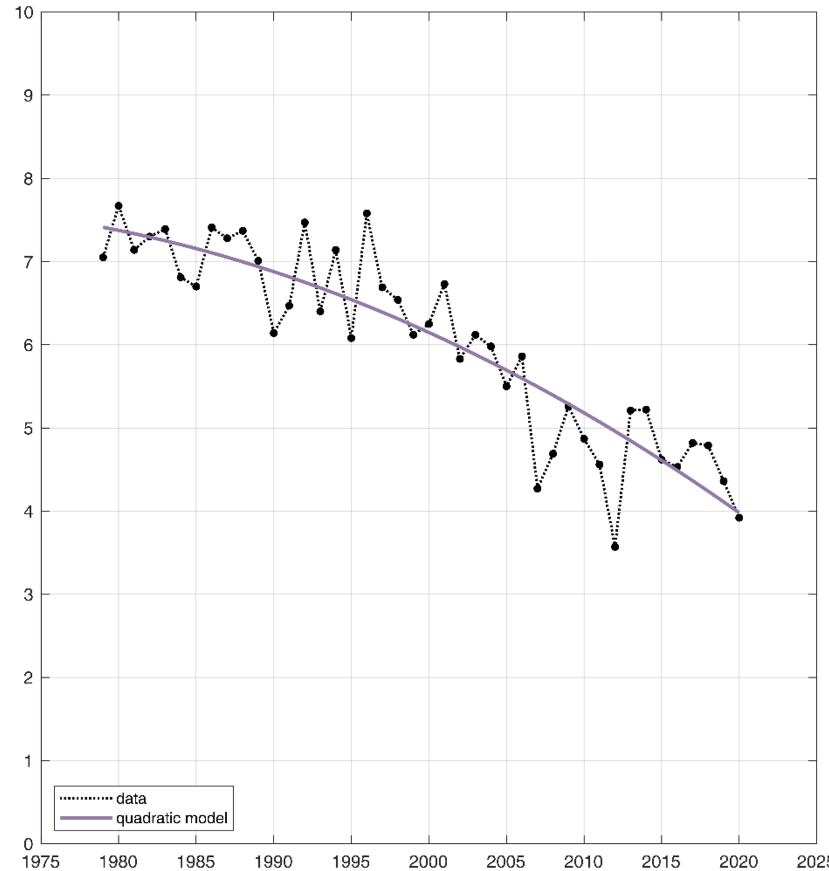
- fitting a higher-order polynomia
- now: three unknowns *A*, *B*, and
- considering the partial derivative
  and setting them to 0, gives a 3

$$E_2(A,B,C) = \sum_k \left(f(x_k) - y_k\right)^2$$

$$\frac{\partial E_2}{\partial A} = 0: \quad \sum_k 2\left(Ax_k^2 + Bx_k + \right.$$

$$\frac{\partial E_2}{\partial B} = 0: \quad \sum_k 2\left(Ax_k^2 + Bx_k + \right.$$

$$\frac{\partial E_2}{\partial C} = 0:$$



$$\begin{pmatrix} \sum_k x_k^4 & \sum_k x_k^3 & \sum_k x_k^2 \\ \sum_k x_k^3 & \sum_k x_k^2 & \sum_k x_k \\ \sum_k x_k^2 & \sum_k x_k & n \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} \sum_k x_k^2 y_k \\ \sum_k x_k y_k \\ \sum_k y_k \end{pmatrix}$$

Other, non-linear Models

Alternative to linear regression:

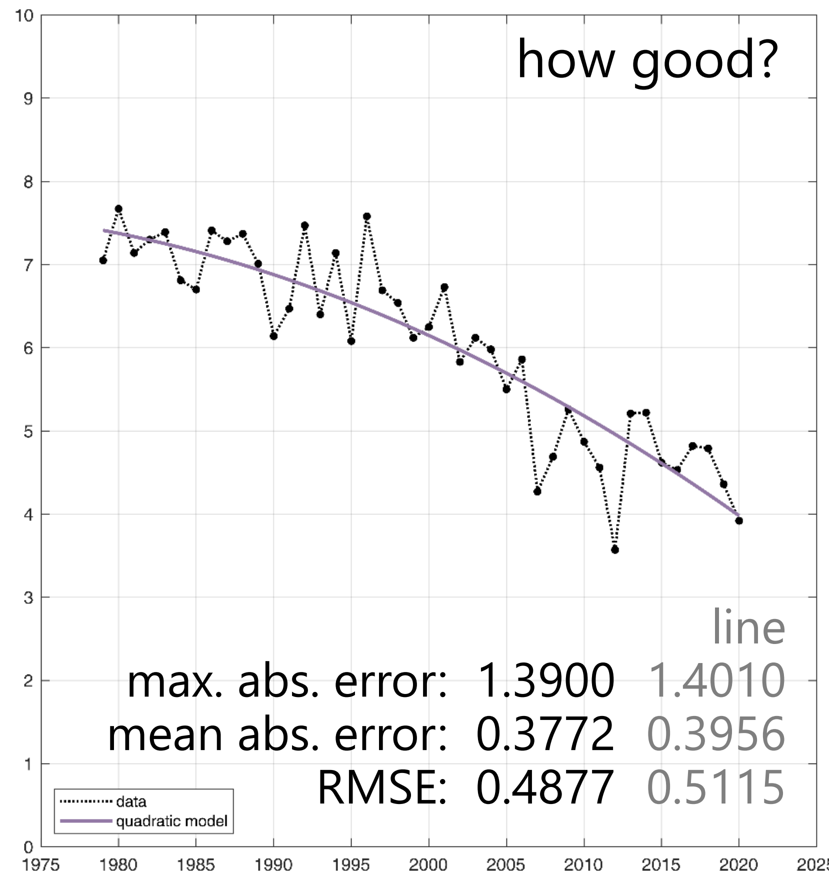
- fitting a higher-order polynomia
- now: three unknowns *A*, *B*, and
- considering the partial derivative and setting them to 0, gives a 3

$$E_2(A,B,C) = \sum_k \left(f(x_k) - y_k\right)^2$$

$$\frac{\partial E_2}{\partial A} = 0 : \quad \sum_k 2\left(Ax_k^2 + Bx_k + \right.$$

$$\frac{\partial E_2}{\partial B} = 0 : \quad \sum_k 2\left(Ax_k^2 + Bx_k + \right.$$

$$\frac{\partial E_2}{\partial C} = 0 :$$



$$\begin{pmatrix} \sum_k x_k^4 & \sum_k x_k^3 & \sum_k x_k^2 \\ \sum_k x_k^3 & \sum_k x_k^2 & \sum_k x_k \\ \sum_k x_k^2 & \sum_k x_k & n \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} \sum_k x_k^2 y_k \\ \sum_k x_k y_k \\ \sum_k y_k \end{pmatrix}$$

Exponential Fit

[see also MATLAB example 3a_C]

Alternative to polynomial fits,

- it can be interesting/relevant
  to approximate the data with an exponential $f(x) = C\, e^{Ax}$
- unfortunately,
  forming the partial derivatives of E_2
  does *not* lead to a system of linear equations

$$C \sum_{k=1}^{n} x_k \exp(2Ax_k) - \sum_{k=1}^{n} x_k y_k \exp(Ax_k) = 0$$

$$C \sum_{k=1}^{n} \exp(Ax_k) - \sum_{k=1}^{n} y_k \exp(Ax_k) = 0$$

- but here:
  a *change of variables* helps:
  - $Y = \ln y \quad (X = x)$
    $B = \ln C$

$$f(x) = y = C \exp(Ax)$$

$$\ln y = \ln(C \exp(Ax)) = \ln C + \ln(\exp(Ax)) = B + Ax \rightarrow Y = AX + B$$

Exponential Fit

[see also MATLAB example 3a_C]

Alternative to polynomial fits,

– it can be interesting/relev
to approximate the data w

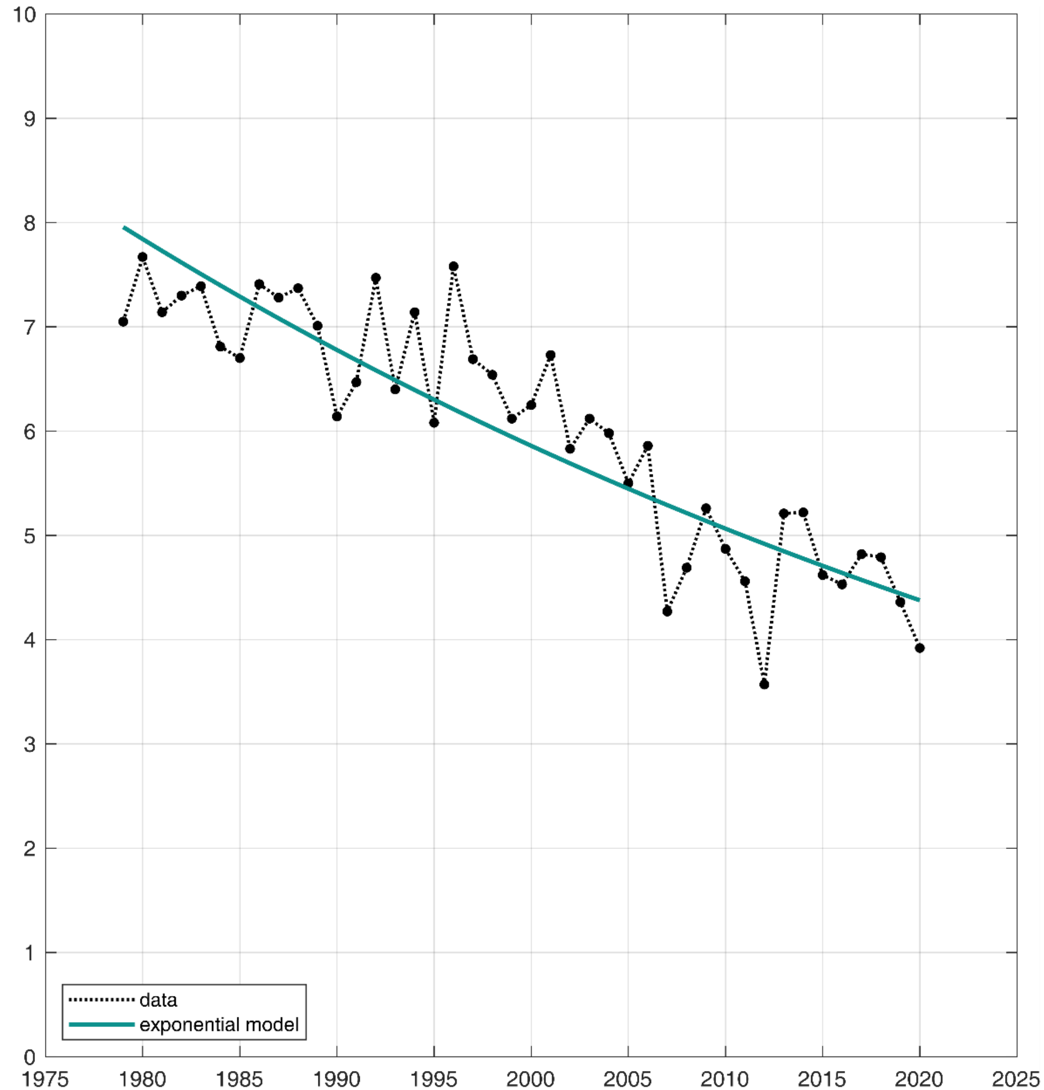
– unfortunately,
forming the partial deriva
does *not* lead to a system

– but here:
a *change of variables*
helps:

- $Y = \ln y \;\; (X = x)$
$B = \ln C$

$$f(x) = y = C\exp(Ax)$$

$$\ln y = \ln(C\exp(Ax)) = \ln C +$$

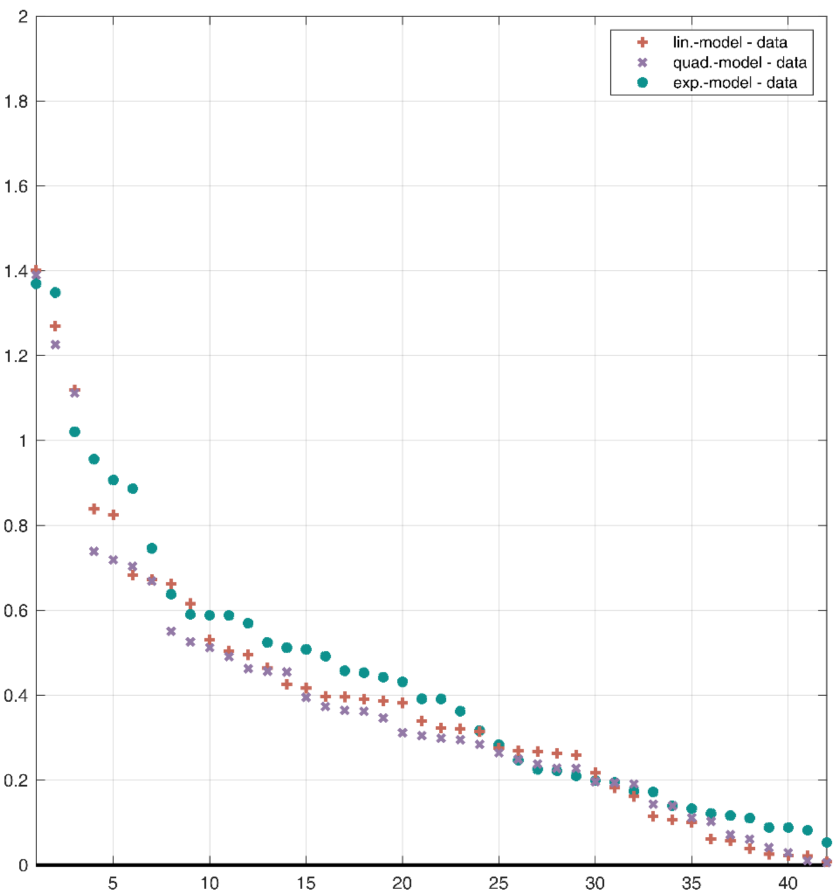


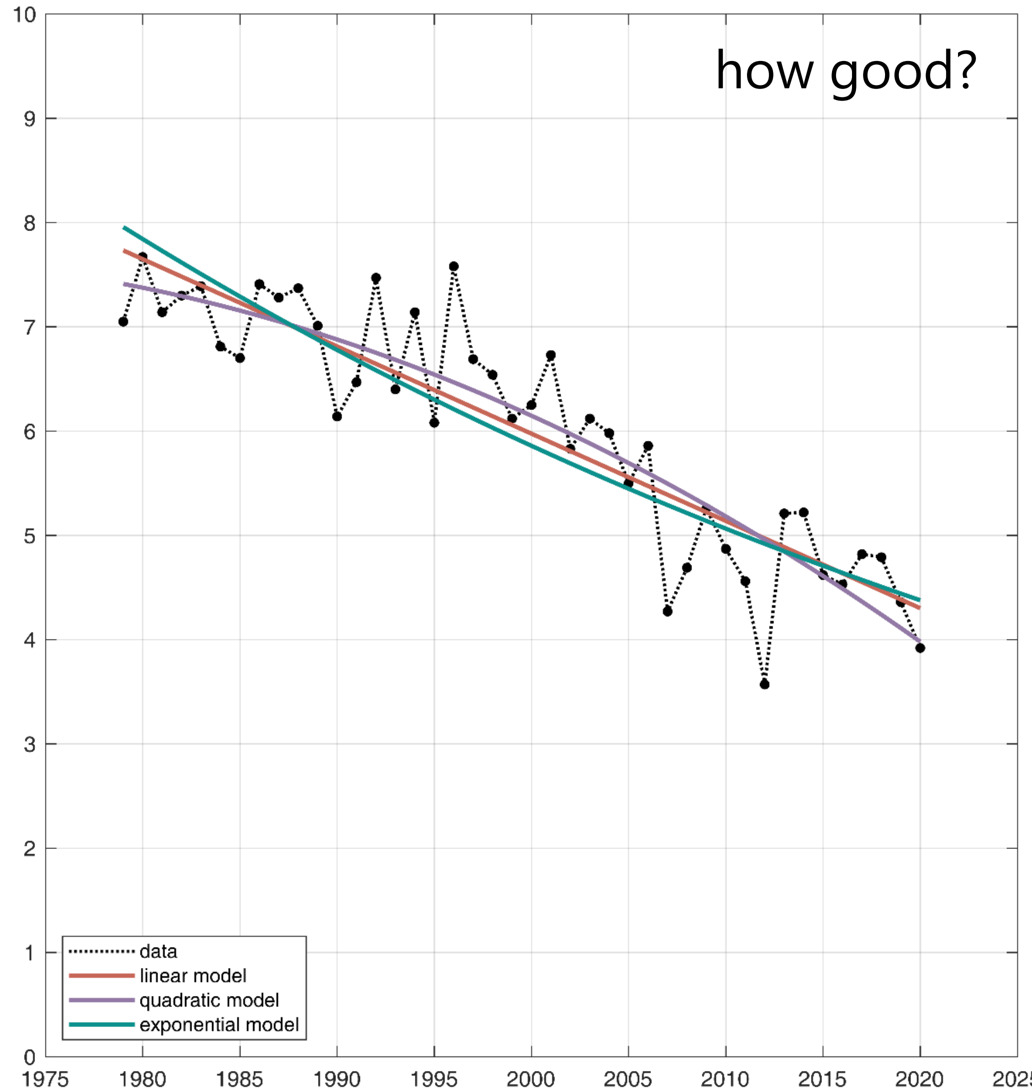
Exponential Fit

[see also MATLAB example 3a_C]

Alternative to polynomial fits,

– it can be interesting/relev
to approximate the data w





Another View on Least-Squares Fitting

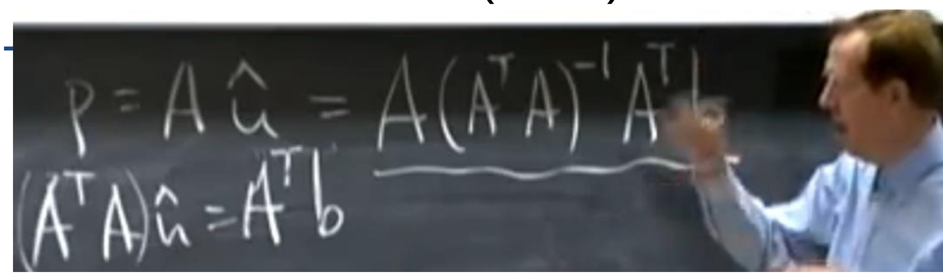
Given overdetermined system $\mathbf{A\,x = b}$ with $\mathbf{A} \in \mathbf{R}^{n \times m}$, $n>m$

- and $\mathbf{A}$ has full (column-)rank,
  i.e., the columns of $\mathbf{A}$ are linearly independent ($\mathbf{A} \leftrightarrow$ basis),
- then $\mathbf{A\,x}$ (for all $\mathbf{x} \in \mathrm{R}^{m \times 1}$)
  generates all vectors $\mathbf{y} \in \mathrm{R}^{n \times 1}$,
  i.e., an m-dimensional subspace $\mathbf{\Gamma}$ of R^n (since m<n),
  for which an exact solution $\mathbf{x}$ (to $\mathbf{A\,x = y}$) exists
- usually, $\mathbf{b} \notin \mathbf{\Gamma}$, i.e., no exact solution $\mathbf{x}$ is found for $\mathbf{A\,x = b}$
- **question**: what's the best $\mathbf{y}_! \in \mathbf{\Gamma}$ (wrt. $\mathbf{A\,x = b}$)?
- **answer** (in the least-squares sense): choose $\mathbf{y}_!$ closest to $\mathbf{b}$!
- the (minimal) error $\mathbf{e}$ is then given as $\mathbf{e} = \mathbf{b} - \mathbf{y}_!$
  and $\mathbf{e}$ is orthogonal to $\mathbf{\Gamma}$!
- thus, $\mathbf{A}^\mathrm{T}\,\mathbf{e} = \mathbf{0} \;\rightarrow\; \mathbf{A}^\mathrm{T}\,\mathbf{e} = \mathbf{A}^\mathrm{T}\,(\mathbf{b} - \mathbf{y}_!) = \mathbf{A}^\mathrm{T}\,(\mathbf{b} - \mathbf{A}\mathbf{x}_!) = \mathbf{0}$
- ergo $(\mathbf{A}^\mathrm{T}\mathbf{A})\,\mathbf{x}_! = \mathbf{A}^\mathrm{T}\,\mathbf{b} \leftarrow$ the ***normal equations*** for best-possible $\mathbf{x}_!$

Another View on Least-Squares Fitting

Given the normal equations $(\mathbf{A}^\mathsf{T}\mathbf{A})\,\mathbf{x}_! = \mathbf{A}^\mathsf{T}\,\mathbf{b}$
for the overdetermined system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ **with** $\mathbf{A} \in \mathbf{R}^{nxm}$, $n>m$

- and full-rank $\mathbf{A}$ (still),
- $(\mathbf{A}^\mathsf{T}\mathbf{A})$ is nicely invertible!
- thus we can find $\mathbf{x}_!$ by $\mathbf{x}_! = \mathbf{A}^+\,\mathbf{b}$ with $\mathbf{A}^+ = (\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\,\mathbf{A}^\mathsf{T}$
  and $\mathbf{A}^+$ being called the pseudo-



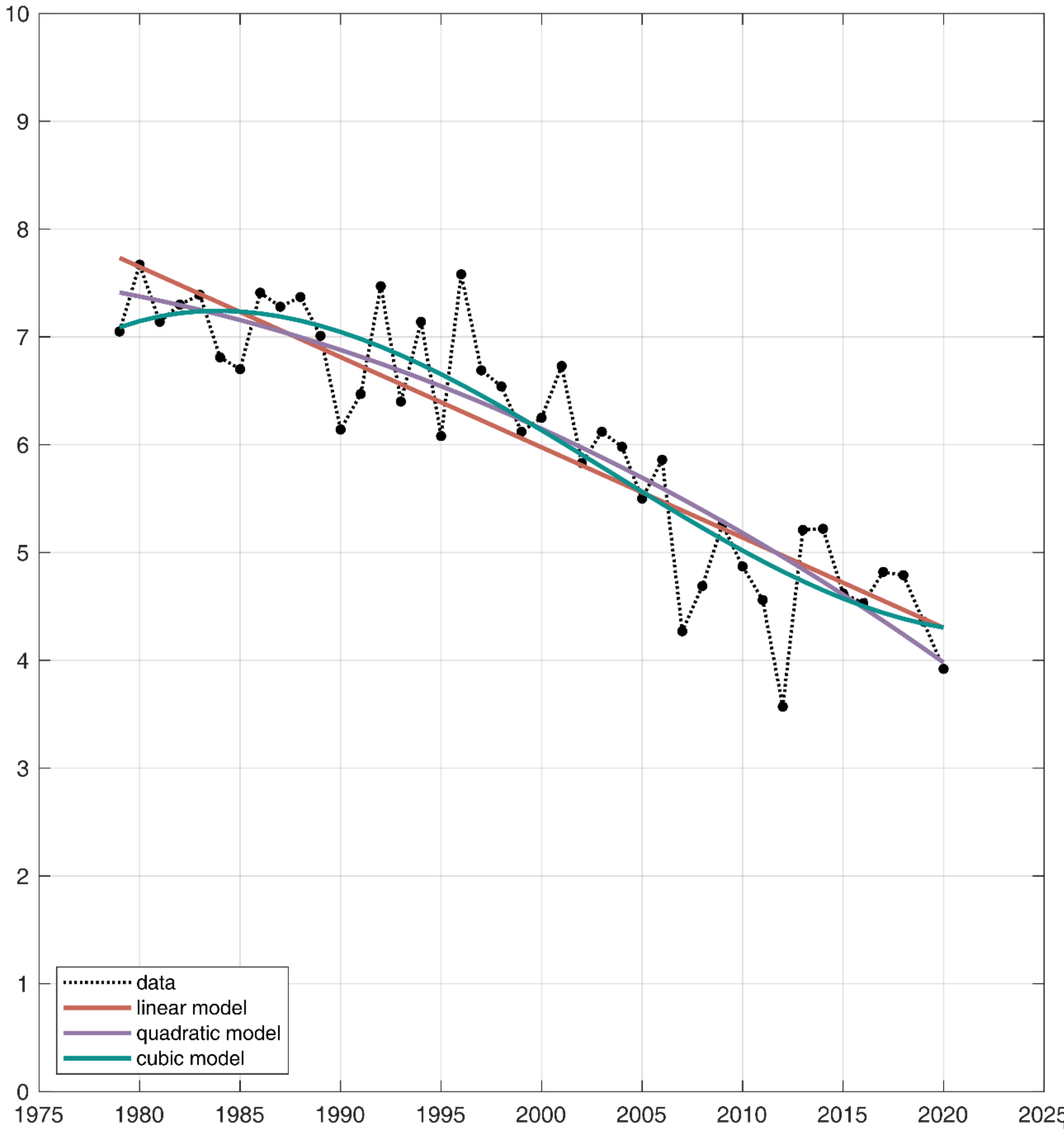
[G. Strang explaining the projection]

Another look at $\mathbf{y}_! = \mathbf{A}\,\mathbf{x}_!$,
i.e., the vector in $\mathbf{\Gamma}$ that is closest to $\mathbf{b}$

- since the error $\mathbf{e} = (\mathbf{b} - \mathbf{y}_!)$ is orthogonal to $\mathbf{\Gamma}$,
- $\mathbf{y}_!$ is the orthogonal projection of $\mathbf{b}$ onto $\mathbf{\Gamma}$
- and $\mathbf{y}_!$ is given by $\mathbf{y}_! = \mathbf{A}\,(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\,\mathbf{A}^\mathsf{T}\,\mathbf{b}$
- thus $\mathbf{P}_\mathbf{\Gamma}$
  is the matrix that projects onto $\mathbf{\Gamma}$, i.e., the column-space of $\mathbf{A}$,
  with $\mathbf{P}_\mathbf{\Gamma} = \mathbf{A}\,(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\,\mathbf{A}^\mathsf{T}$

[see also MATLAB example 3a_D]

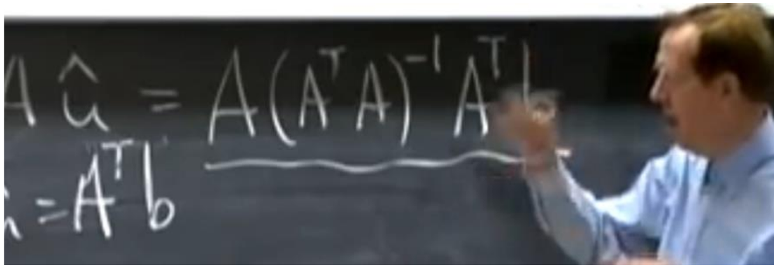
Another View on Least-Squares Fitting



$= \mathbf{A}^{\mathsf{T}}\,\mathbf{b}$

$: \mathbf{b}$ **with** $\mathbf{A} \in \mathbf{R}^{nxm}$, $n>m$

with $\mathbf{A}^{+} = (\mathbf{A}^{\mathsf{T}}\mathbf{A})^{-1}\,\mathbf{A}^{\mathsf{T}}$



[G. Strang explaining the projection]

al to **Γ**,

to **Γ**

b

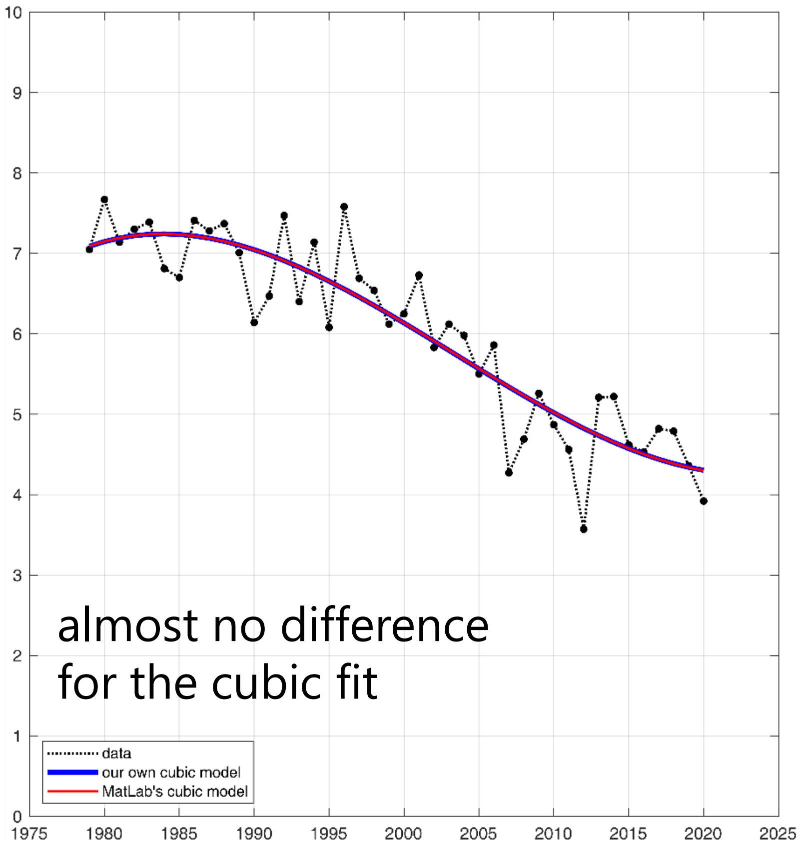
he column-space of **A**,

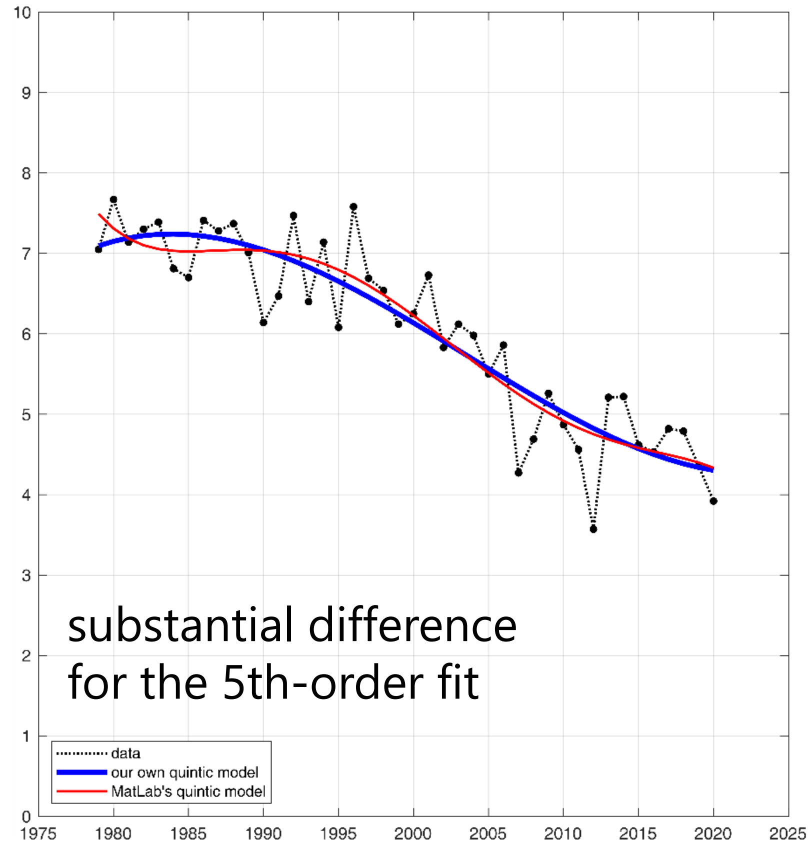
[see also MATLAB example 3a_D]

Numerical Issues

When attempting to compute higher-order fits:

- likely, the equation system becomes numerically ill-conditioned (think of the high powers of x_k in the normal equations!)
- rescaling of x, for ex., to the unit interval, can help



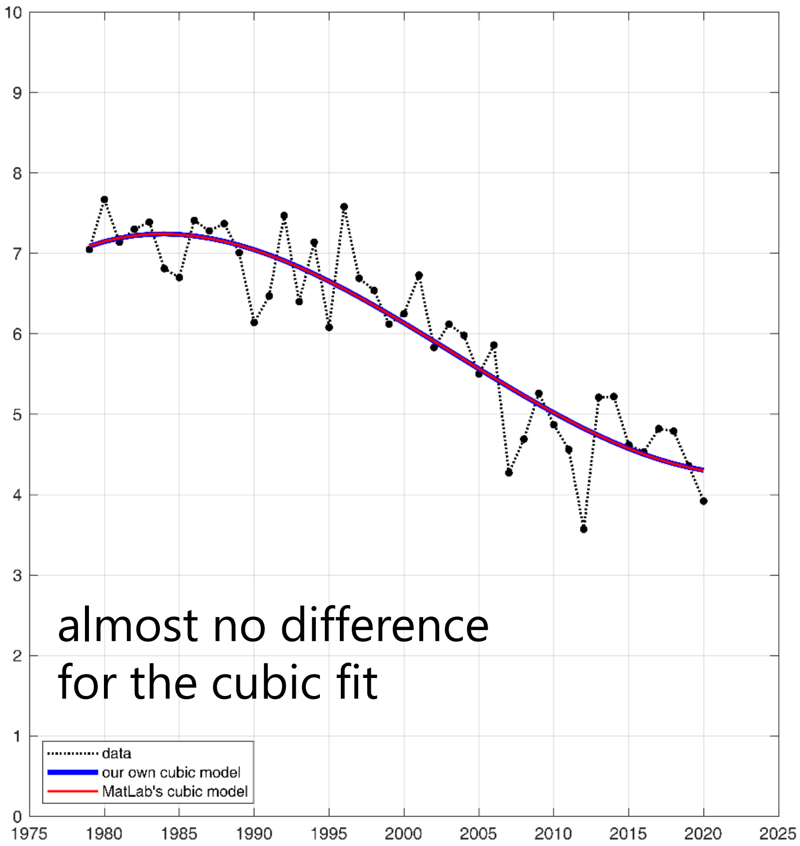


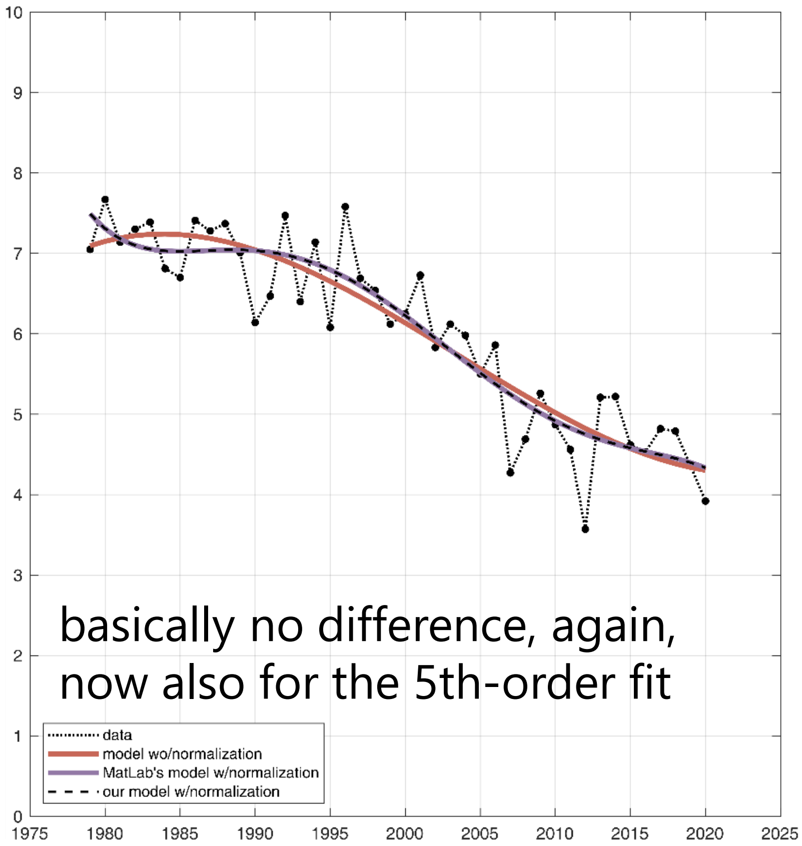
Numerical Issues

When attempting to compute higher-order fits:

– likely, the equation system becomes numerically ill-conditioned (think of the high powers of x_k in the normal equations!)

– rescaling of x, for ex., to the unit interval, can help



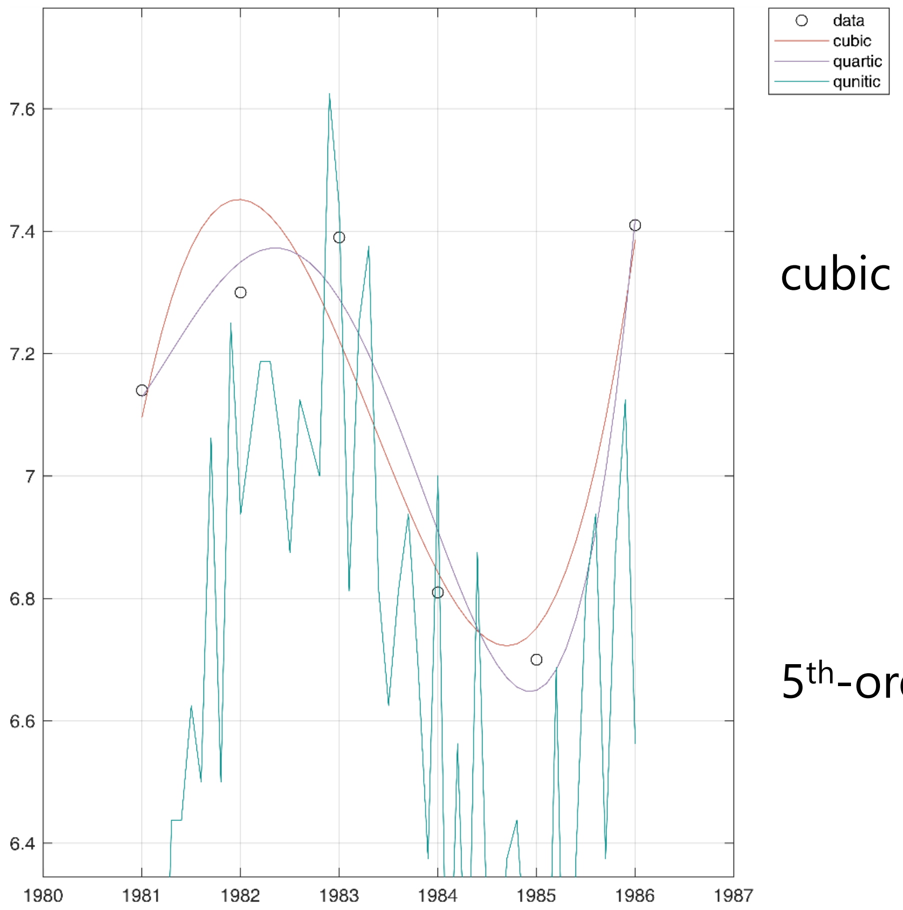


Issues of Higher-order Fits

[see also MATLAB example 3a_E]

For non-tiny datasets:

- low-order fitting is usually fine (regression line, quadratic fit, ...)
- higher-order fits lead to numerical issues—below with polyfit()...



cubic and 4th-order fit (OK)

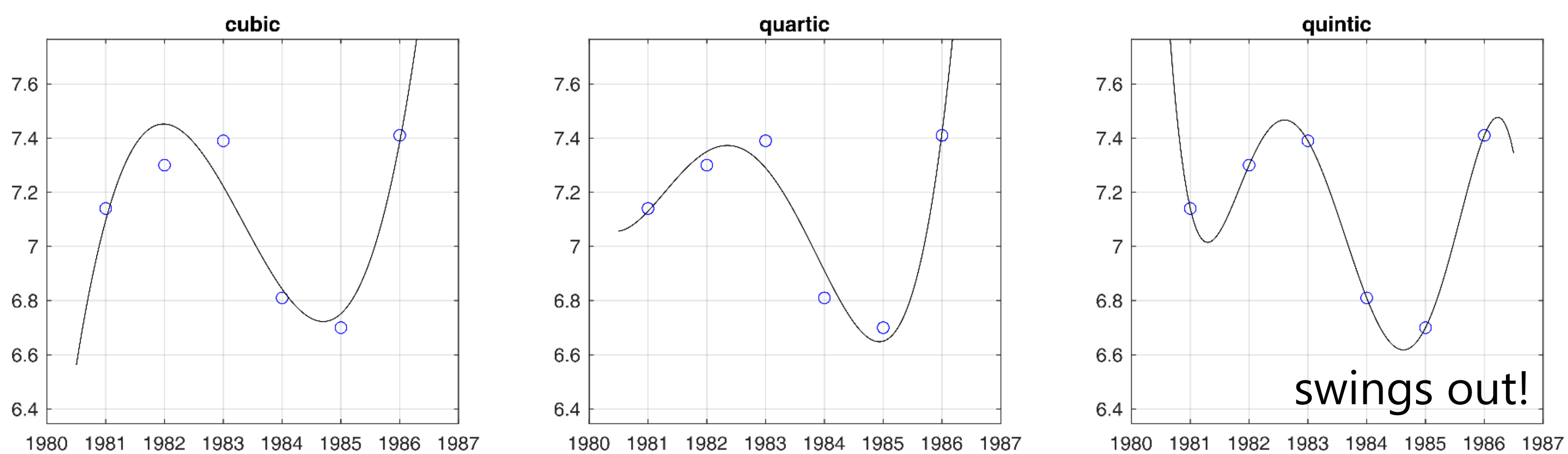
5th-order fit (not OK)

Issues of Higher-order Fits

[see also MATLAB example 3a_E]

For non-tiny datasets:

- low-order fitting is usually fine (regression line, quadratic fit, …)
- higher-order fits lead to numerical issues
  - centering & normalization of the abscissa helps (a bit)…
- still:
  - the standard approaches to polynomial fitting like polyfit() run into numerical problems fairly soon
  - higher-order polynomial models tend to oscillate at the ends of the abscissa-interval (on the left, on the right)



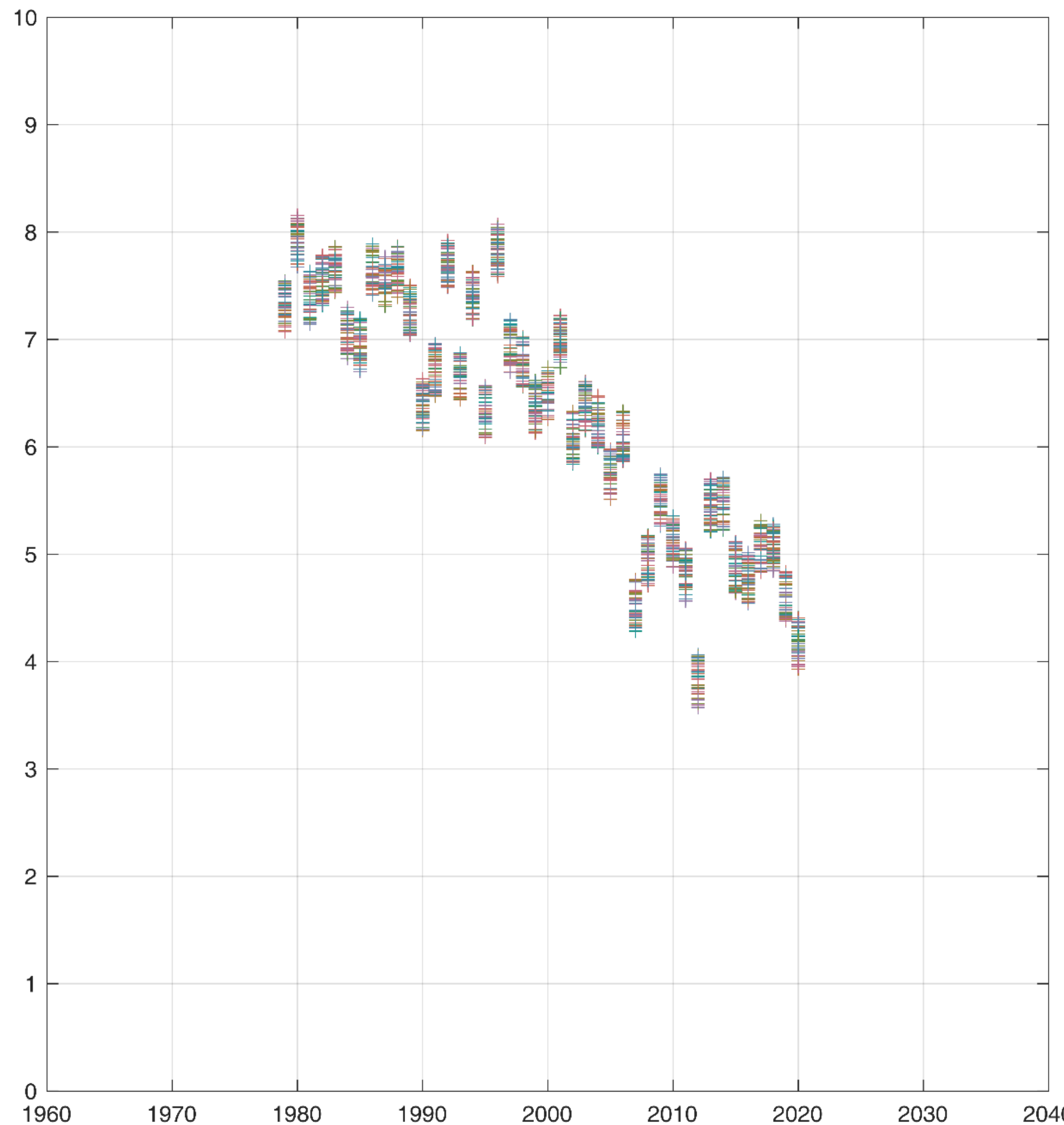
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

– slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise

– we also show a bit of past and future to test prediction



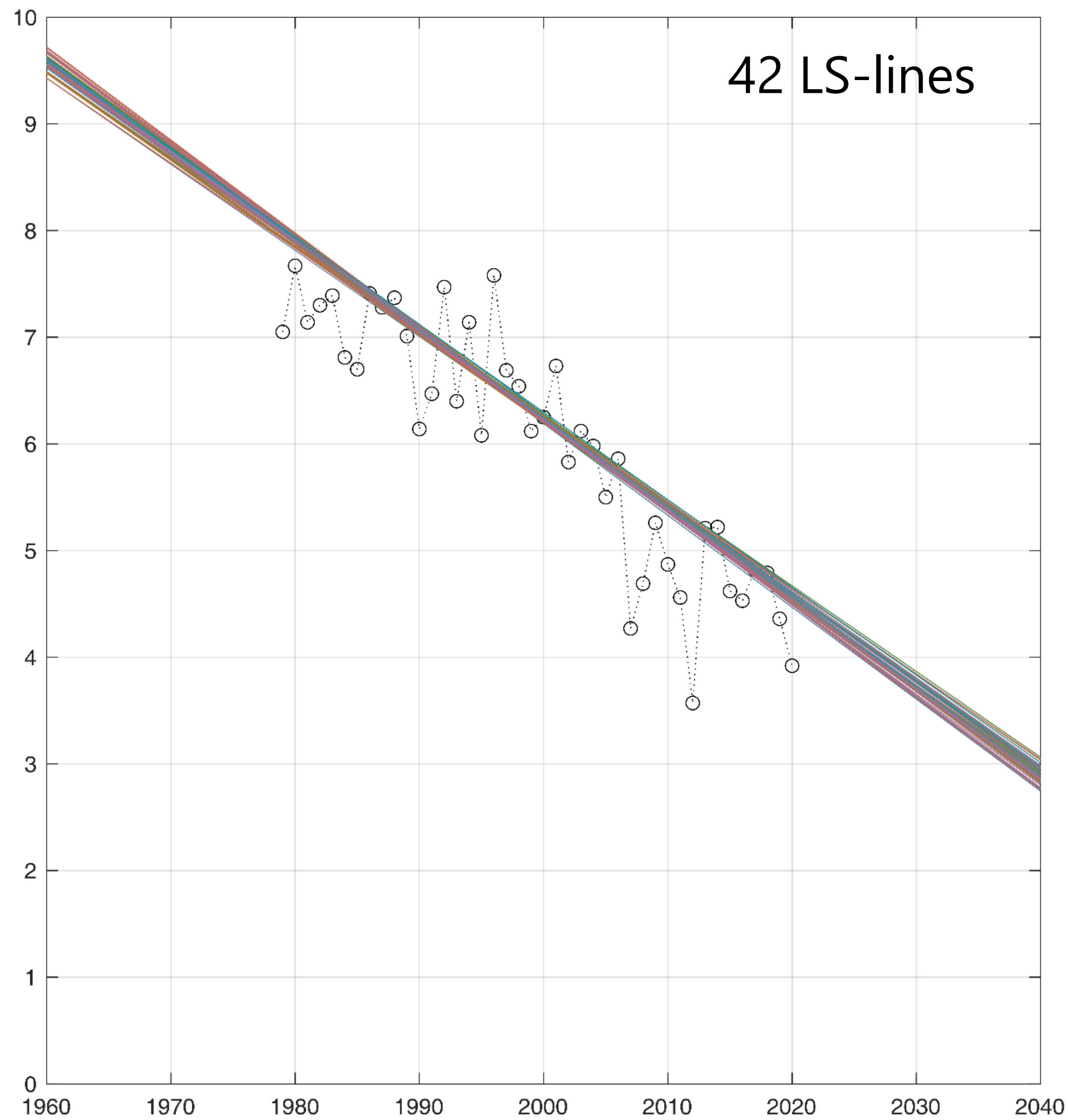
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit of past and future to test prediction

LS-lines appear quite stable and allow (some careful) prediction



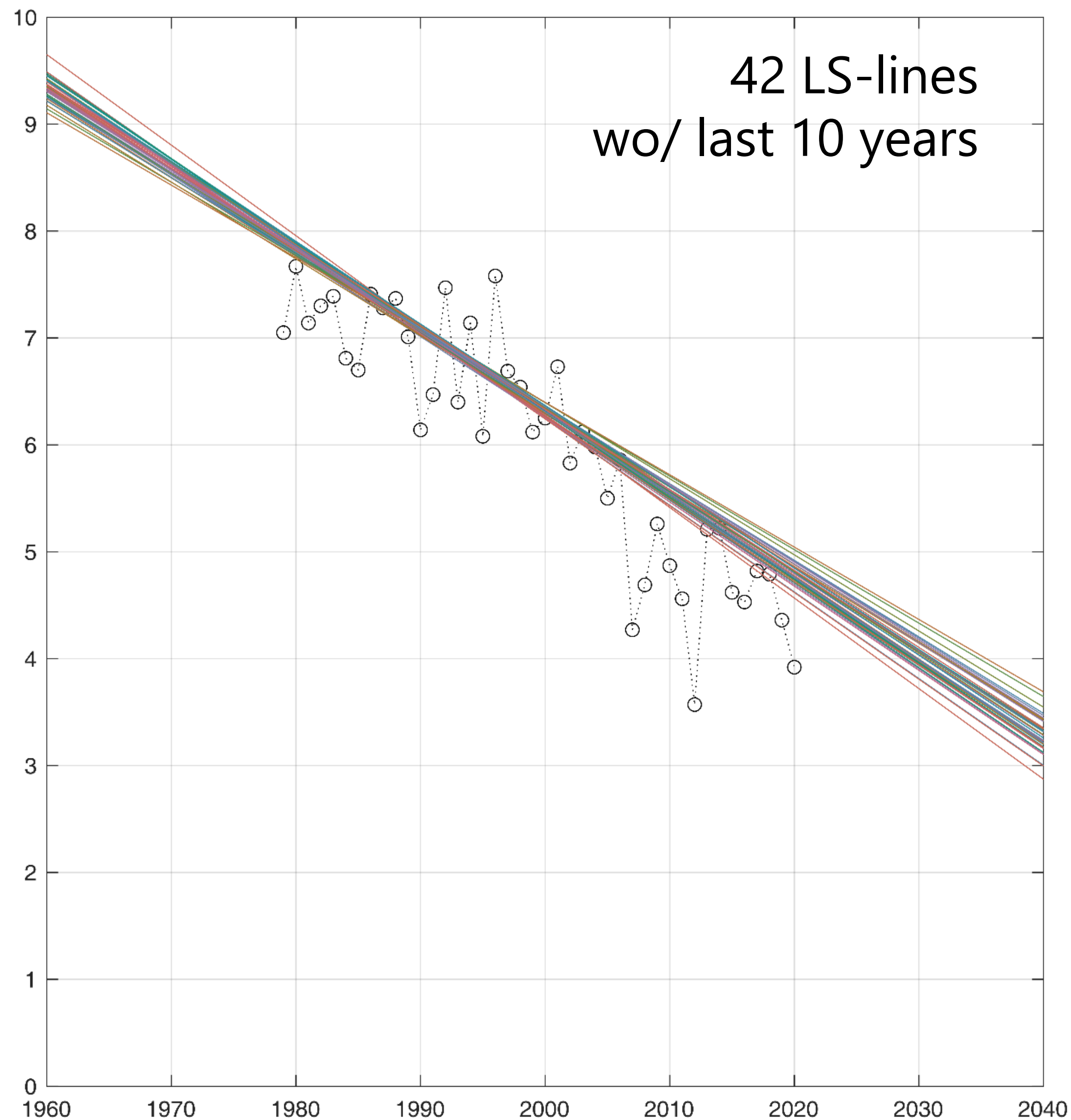
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations
  of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit
  of past and future
  to test prediction

linear prediction
based on the data
up to 10 years ago



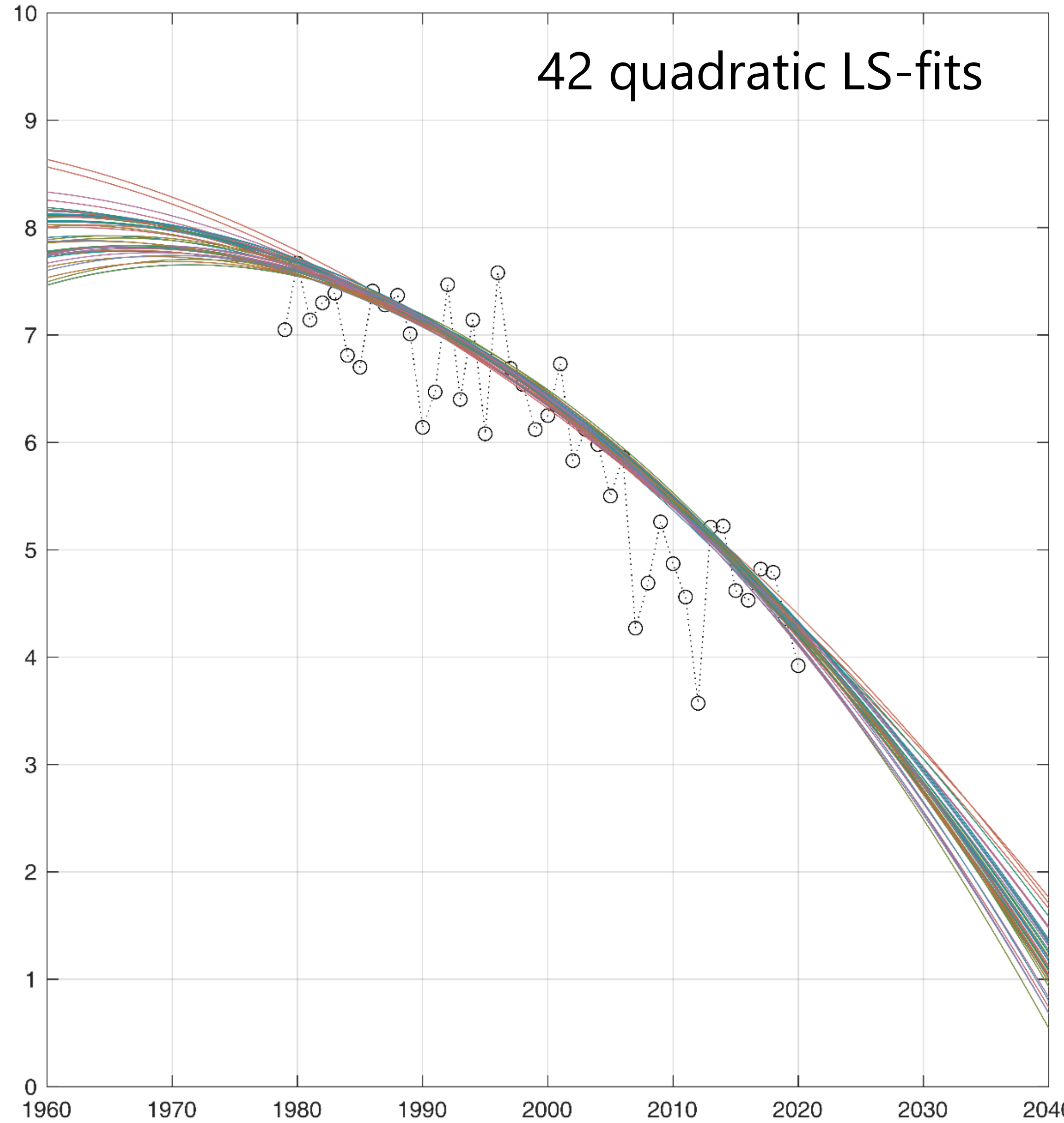
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit of past and future to test prediction

quadratic fits are less stable and long-term prediction seems questionable



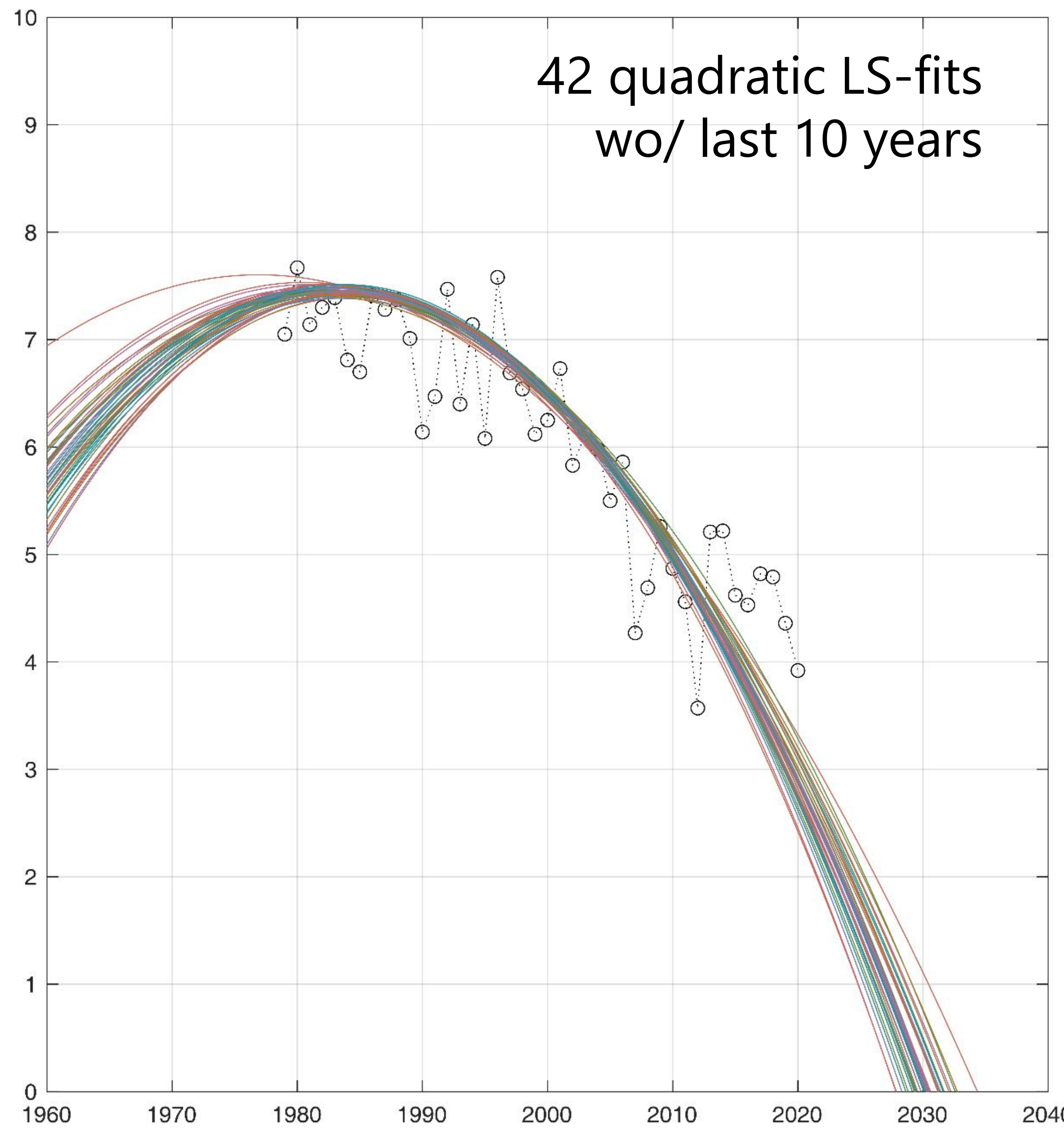
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit of past and future to test prediction

quadratic prediction based on the data up to 10 years ago



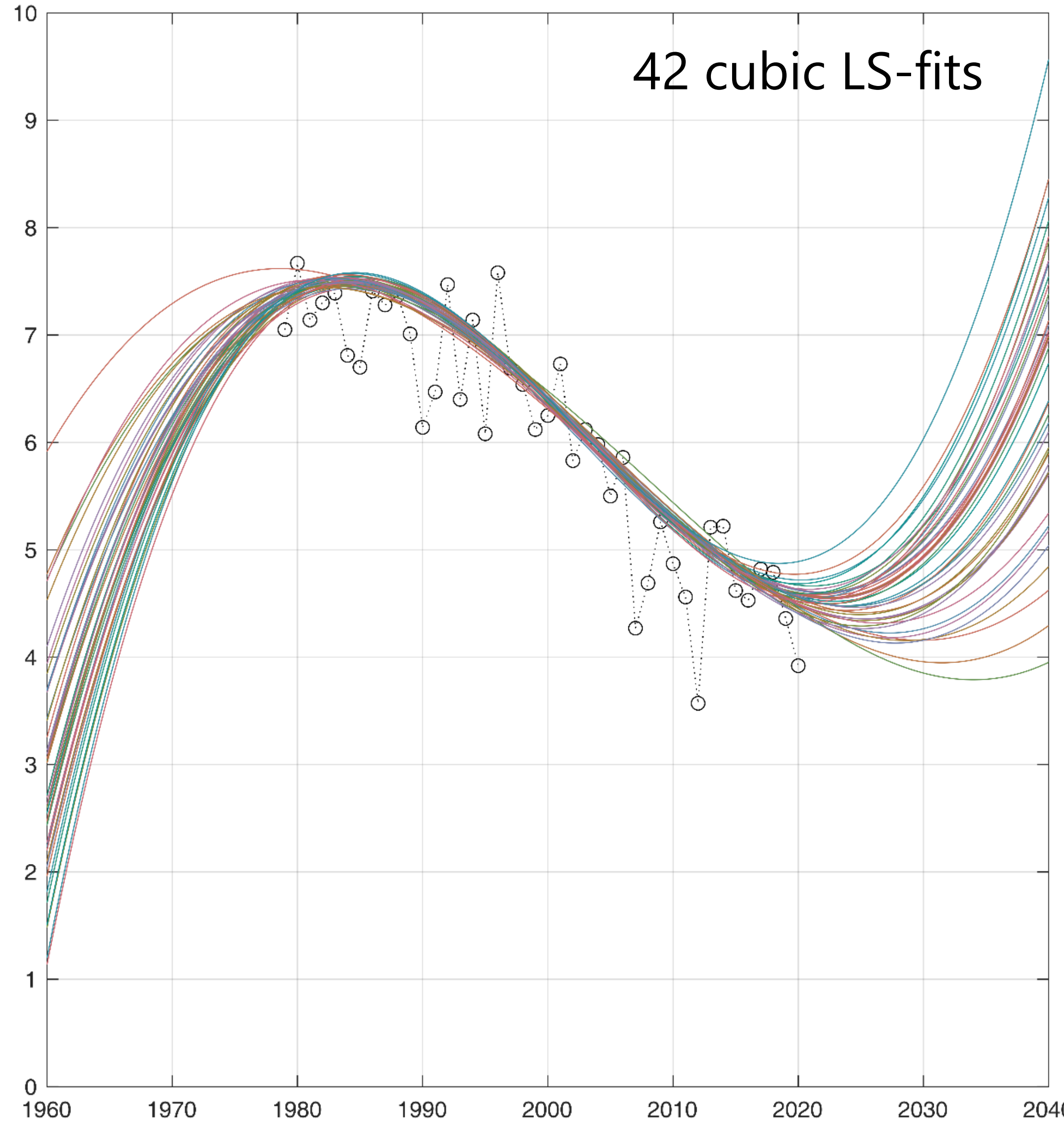
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit of past and future to test prediction

cubic fits are spreading a lot and provide no real information 20 years into the future



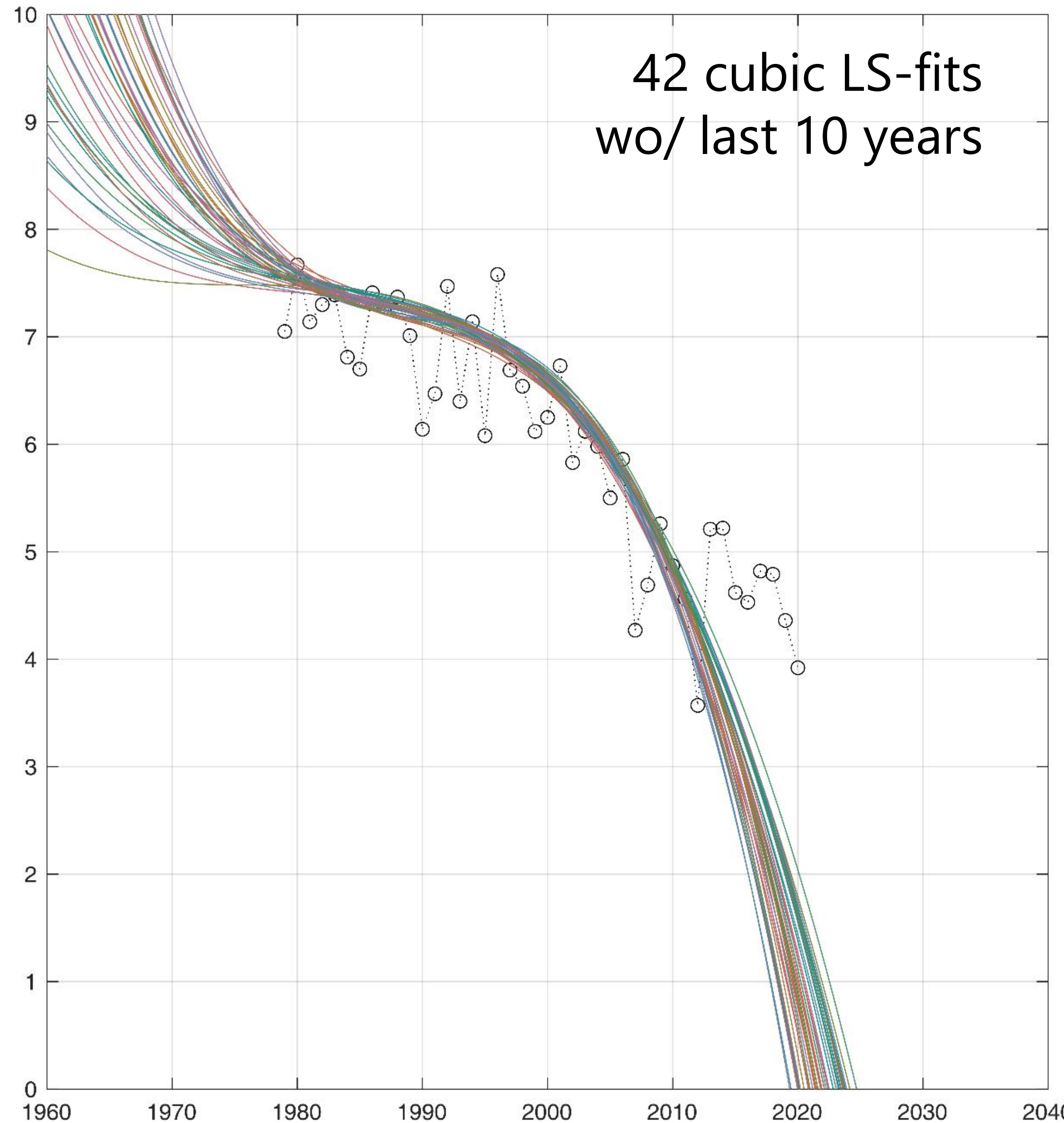
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit of past and future to test prediction

cubic prediction based on the data up to 10 years ago



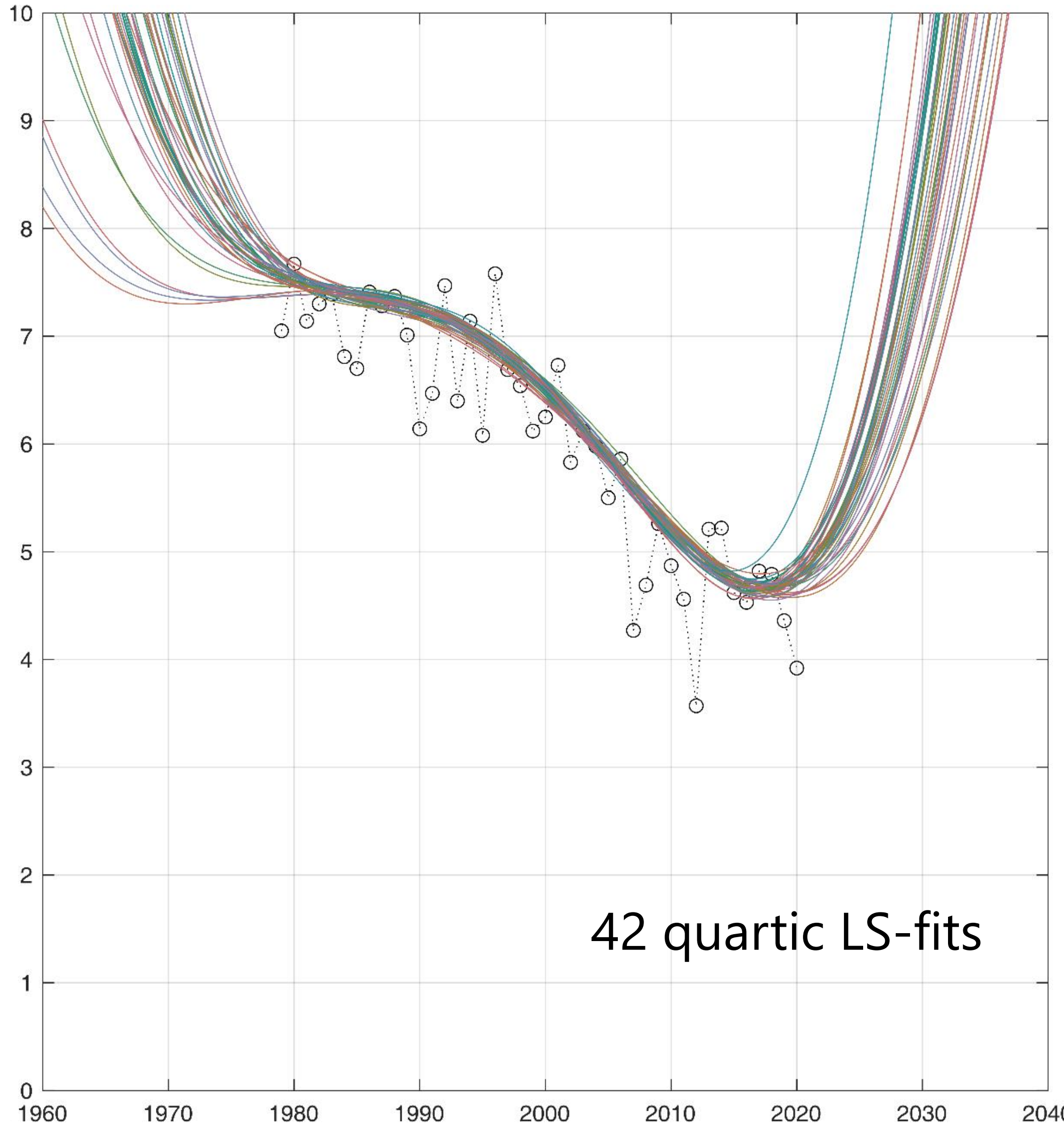
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit of past and future to test prediction

quartic fits are, of course, even less stable and thus also less suitable for prediction



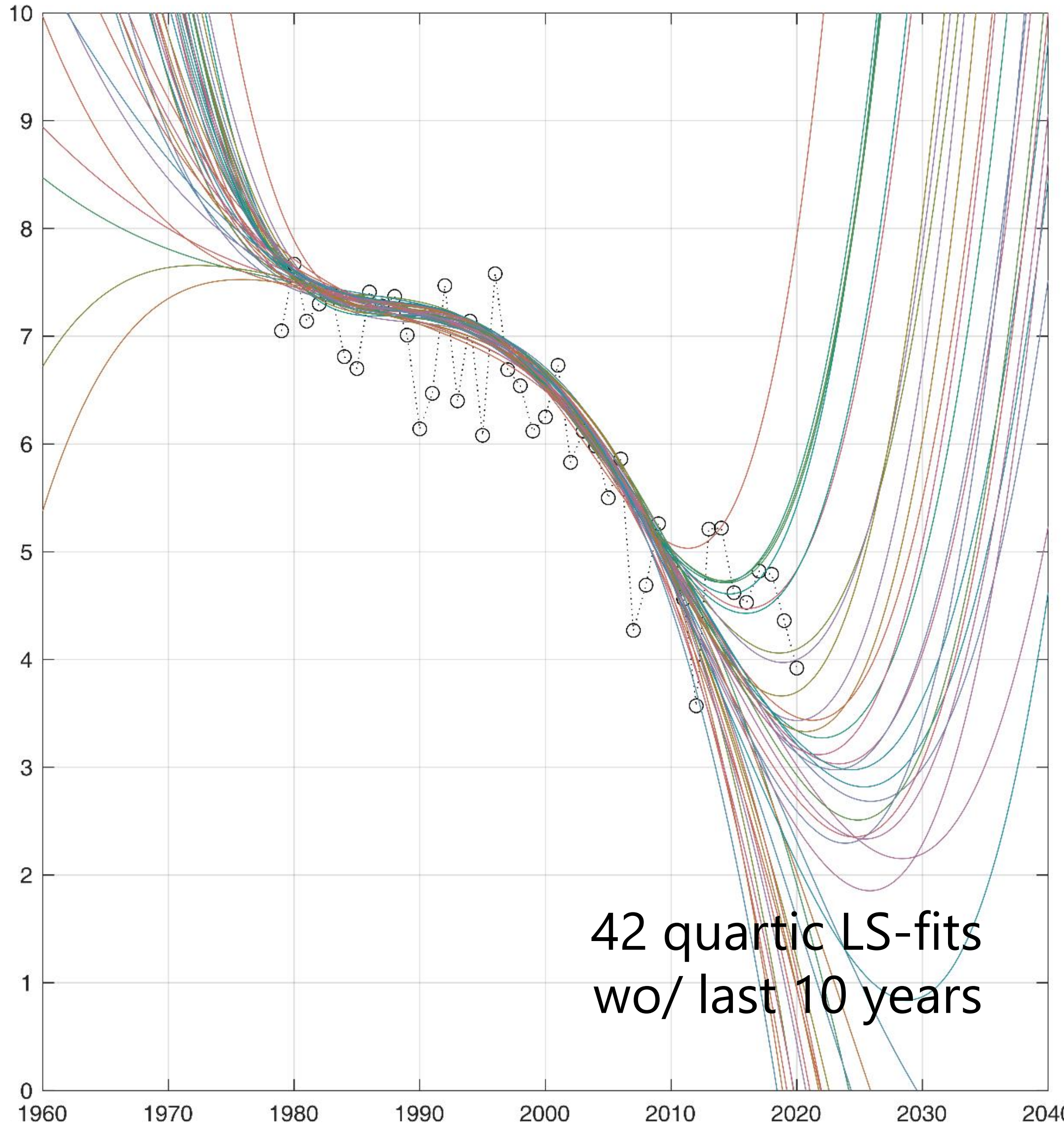
Issues of Global Polynomial Models

[see also MATLAB example 3a_F]

Stability of solution:

- slight variations of the sea ice data:
  - leave out one point
  - add 5% of noise
- we also show a bit of past and future to test prediction

quartic prediction based on the data up to 10 years ago



Outlook

Next:

- piece-wise modeling with splines

Reading and Related Material

In the book:

- chapter 3 (on curve fitting) and related parts

On Wikipedia:

- many good pages