

# Scientific Computing 2013

Maastricht Science Program

## Week 3

Frans Oliehoek  
<frans.oliehoek@maastrichtuniversity.nl>

# Recap

- Matlab...!
- Advanced calculator
  - operator priorities, variable names, matlab functions
- Using scripts
- Example of data reductions using PCA
- Floating point numbers

# This Lecture

- Vectors & Matrices in Matlab
  - creating, indexing, using functions
- Given data: figure out how variables relate.
  - E.g., given medical symptoms or measurements, what is the probability of some disease?
- Estimating functions from a number of data points.
  - Interpolation, Least Squares Regression

NOTE: It is a lot...!

# Matrices & Vectors

# Motivation

- LA is the basis of **many** methods in science
- For us:
  - Important to solve systems of linear equations

$$a_1 x_1 + a_2 x_2 + \dots = c$$

$$a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n = c_1$$

$$a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n = c_2$$

...

$$a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n = c_m$$

- Arise in many problems, e.g.:
  - Identifying gas mixture from peaks in spectrum
  - fitting a line to data.

# Motivation

- LA is the basis of **many** methods in science

- $x_j$  - the amount of gas of type  $j$
- $a_{ij}$  - how much a gas of type  $j$  contributes to wavelength  $i$
- $c_i$  - the height of the peak of wavelength  $i$

Systems of linear equations

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = c_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = c_2$$

...

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = c_m$$

- Arise in many problems, e.g.:
  - Identifying gas mixture from peaks in spectrum
  - fitting a line to data.

# Linear System of Equations

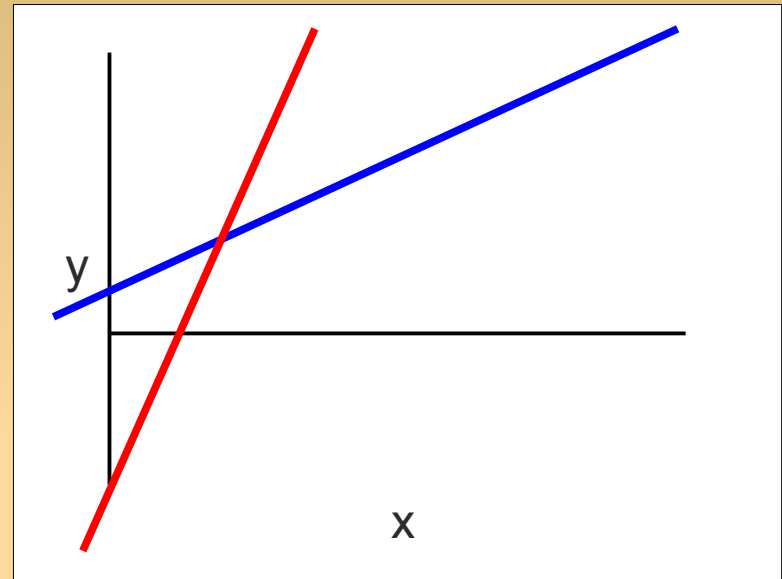
- Example

$$y = 0.5x + 1$$

$$y = 2x - 3$$

- Infinitely many, one, or no solution

- matrices make these easy work with



Another reason to care about matrices and vectors:

they can make complex problems easy to write down!

# Matrices

- A **matrix** with
  - m rows,
  - n columnsis a collection of numbers
  - represented as a table

$$A = \begin{bmatrix} 3 & -2 & 6 \\ 5 & 2 & -8 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 54 & 6 \\ 75 & 24 & 81 \\ 25 & 5 & 435 \end{bmatrix}$$

- A **vector** is a matrix that is
  - 1 row (row vector), or
  - 1 column (column vector)

$$v = [3 \quad -2 \quad 6]$$

$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix}$$



# Matrices

- A **matrix** with

- m rows,
- n columns

is a collection of numbers

- represented as a table

- A **vector** is a matrix that is

- 1 row (row vector), or
- 1 column (column vector)

$$A = \begin{bmatrix} 3 & -2 & 6 \\ 5 & 2 & -8 \end{bmatrix}$$

```
octave:1> A = [3, -2, 6; 5, 2, -8]
```

```
A =
```

$$\begin{matrix} 3 & -2 & 6 \\ 5 & 2 & -8 \end{matrix}$$

$$= \begin{bmatrix} 5 & 54 & 6 \\ 75 & 24 & 81 \\ 25 & 5 & 435 \end{bmatrix}$$

```
octave:2> w = [5;75;25]
```

```
w =
```

$$\begin{matrix} 5 \\ 75 \\ 25 \end{matrix}$$

$$v = \begin{bmatrix} 3 & -2 & 6 \end{bmatrix}$$

$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix}$$

# Matrices

- A **matrix** with

- m rows,
- n columns

is a collection of numbers

- represented as a table

$$A = \begin{bmatrix} 3 & -2 & 6 \\ 5 & 2 & -8 \end{bmatrix}$$

```
octave:1> A = [3, -2, 6; 5, 2, -8]
A =
```

```
3 -2 6
5 2 -8
```

```
octave:2> w = [5;75;25]
w =
```

```
5
75
25
```

- A **vector** is a matrix that is

- 1 row (row vector), or
- 1 column (column vector)

```
octave:3> a1 = [4:8]
a1 =
```

```
4 5 6 7 8
```

```
octave:4> a2 = [4:2:8]
a2 =
```

```
4 6 8
```

# Some Special Matrices

- **Square** matrix:  $m=n$
- **Identity** matrix - 'eye(3)'
- **Zero** matrix – 'zeros(m,n)'

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Types: diagonal, triangular (upper & lower)

$$D = \begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix} \quad TU = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix} \quad TL = \begin{bmatrix} * & 0 & 0 \\ * & * & 0 \\ * & * & * \end{bmatrix}$$

- '\*' denotes any number

# Operations on Vectors - 1

- We can perform operations on them!
  - First: vectors. Next: generalization to matrices.
- **Transpose:** convert row  $\leftrightarrow$  column vector

$$v = [3 \quad -2 \quad 6] \qquad v^T = \begin{bmatrix} 3 \\ -2 \\ 6 \end{bmatrix}$$
$$w = \begin{bmatrix} 5 \\ 75 \\ 25 \end{bmatrix} \qquad w^T = [5 \quad 75 \quad 25]$$

# Operations on Vectors - 1

- We can perform operations on them!

- First: vectors. Next generalization to matrices.

```
octave:9> a = [1,4,-2498,12.4]
```

```
a =  
    1.0000    4.0000 -2498.0000   12.4000
```

- **Transpose:** convert row  $\leftrightarrow$  column vector

```
octave:10> a'  
ans =
```

```
    1.0000  
    4.0000  
 -2498.0000  
   12.4000  
v = [3  
     4  
    -2498  
     12.4]
```

$$v^T = \begin{bmatrix} 3 \\ -2 \\ 6 \end{bmatrix}$$

```
octave:11> a''  
ans =
```

```
    5  
   75  
   25  
w = [1.0000  
     4.0000  
    -2498.0000  
     12.4000]
```

$$w^T = [5 \quad 75 \quad 25] \quad 12.4000$$

# Operations on Vectors - 2

- Sum  $[1 \ 2 \ 3] + [10 \ 20 \ 30] = [11 \ 22 \ 33]$
- Product with scalar  $5 * [1 \ 2 \ 3] = [5 \ 10 \ 15]$
- Inner product (also: 'scalar product' or 'dot product')

$$(v, w) = v^T w = \sum_{k=1}^n v_k w_k$$

# Operations on Vectors - 2

- Sum  $[1 \ 2 \ 3] + [10 \ 20 \ 30] = [11 \ 22 \ 33]$
- Product with scalar  $5 * [1 \ 2 \ 3] = [5 \ 10 \ 15]$

- Inner product (also: 'scalar product' or 'dot product')

$$(v, w) = v^T w = \sum_{k=1}^n v_k w_k$$

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, w = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$

$$[1 \ 2 \ 3] \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} = 1 * 10 + 2 * 20 + 3 * 30 = 10 + 40 + 90 = 140$$

# Operations on Vectors - 2

- Sum  $[1 \ 2 \ 3] + [10 \ 20 \ 30] = [11 \ 22 \ 33]$

- Product with scalar

```
octave:4> a = [1;2;3]
a =
     1
     2
     3
```

- Inner product (also: 'scalar product' or 'dot product')

$$(v, w) = v^T w = \sum_{k=1}^n v_k w_k$$

$$[1 \ 2 \ 3] \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} = 1*10 + 2*20 + 3*30 = 10 + 40 + 90 = 140$$

```
octave:5> b = [4;5;6]
b =
     4
     5
     6
```

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, w = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$

```
octave:6> dot(a,b)
ans = 32
octave:7> a'*b
ans = 32
```



# Vector Indexing

- Retrieve parts of vectors

```
octave:12> a = [10, 20, 30, 40, 50, 60, 70]
```

```
a =
```

```
    10    20    30    40    50    60    70
```

```
octave:13> a(3)
```

```
ans = 30
```

```
octave:14> a([2,4])
```

```
ans =
```

```
    20    40
```

```
octave:16> a([4:end])
```

```
ans =
```

```
    40    50    60    70
```

# Vector Indexing

- Retrieve parts of vectors

```
octave:12> a = [10, 20, 30, 40, 50, 60, 70]
```

```
a =
```

```
    10    20    30    40    50    60    70
```

```
octave:13> a(3)
```

```
ans = 30
```

```
octave:14> a([2,4])
```

```
ans =
```


```
    20    40
```

```
octave:16> a([4:end])
```


```
ans =
```

```
    40    50    60    70
```

indexing with  
another vector



special 'end'  
index



# Operations on Matrices - 1

- Now matrices!
- **Transpose:**
  - convert each row  $\rightarrow$  column vector  
(or convert each column  $\rightarrow$  row vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 10 & 100 \\ 2 & 20 & 200 \\ 3 & 30 & 300 \end{bmatrix}$$

# Operations on Matrices - 1

- Now matrices!
- **Transpose:**
  - convert each row  $\rightarrow$  column vector  
(or convert each column  $\rightarrow$  row vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 10 & 100 \\ 2 & 20 & 200 \\ 3 & 30 & 300 \end{bmatrix}$$

# Operations on Matrices - 1

- Now matrices!
- **Transpose:**
  - convert each row  $\rightarrow$  column vector  
(or convert each column  $\rightarrow$  row vector)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 10 & 100 \\ 2 & 20 & 200 \\ 3 & 30 & 300 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{bmatrix}$$

$$B^T = \begin{bmatrix} 1 & 10 \\ 2 & 20 \\ 3 & 30 \end{bmatrix}$$

# Operations on Matrices - 2

- **Sum and product with scalar:** pretty much the same

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{bmatrix} = \begin{bmatrix} 11 & 22 & 33 \\ 44 & 55 & 66 \end{bmatrix}$$

$$5 * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 5 & 10 & 15 \\ 20 & 25 & 30 \end{bmatrix}$$

# Matrix Product

- Inner product → **Matrix product**

$$C = AB$$

- $C = m \times n, \quad A = m \times p, \quad B = p \times n,$

- Each entry of C is an inner product:  $c_{ij} = r_i^A \cdot c_j^B$

$$\begin{bmatrix} \dots & \dots & \dots \\ \mathbf{190} & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ \mathbf{30} & \mathbf{40} \\ 50 & 60 \end{bmatrix} \begin{bmatrix} \mathbf{1} & 2 & 3 \\ \mathbf{4} & 5 & 6 \end{bmatrix}$$

# Matrix Product

- Inner product → **Matrix product**

$$C = AB$$

- $C = m \times n$ ,  $A = m \times p$ ,  $B = p \times n$ ,

- Each entry of  $C$  is an inner product:

$$\begin{bmatrix} \dots & \dots & \dots \\ 190 & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```

octave:22> A = [10, 20; 30, 40; 50, 60]
A =
    10    20
    30    40
    50    60
octave:23> B = [1,2,3;4,5,6]
B =
     1     2     3
     4     5     6
octave:24> A*B
ans =
    90    120    150
   190    260    330
   290    400    510
    
```

$$C_{ij} = r_i \cdot c_j$$



# Matrix Product

- Inner product → Matrix product

```
octave:22> A = [10, 20; 30, 40; 50, 60]
```

```
A =
```

```
10 20
```

```
30 40
```

```
50 60
```

$C = AB$ ,  $A = m \times p$ ,  $B = p \times n$ ,

Each entry of  $C$  is an inner product:  
Btrans =

Matrix size is important

```
1 4
```

```
2 5
```

```
3 6
```

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
octave:26> A*Btrans
```

```
error: operator *: nonconformant arguments (op1 is 3x2, op2 is 3x2)
```

# Matrix-Vector Product

- Matrix-vector product is just a (frequently occurring) special case:

$$Ab = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} b_1 \\ \dots \\ b_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \dots \\ c_m \end{bmatrix}$$

# Matrix-Vector Product

- Also represents a system of equations!

$$Ax = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \dots \\ c_m \end{bmatrix}$$

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = c_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = c_2$$

...

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = c_m$$

# Approximation of Data and Functions

# Approximations of Functions

- Function approximation:  
Replace a function by a simpler one
- Reasons:
  - Integration: replace a complex function with one that is easy to integrate.
  - Function may be very complex: e.g. result of simulation.
  - Function may be unknown...

# “Approximation of Data”

- 'the function unknown'
  - it is only known at certain points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$
  - but we also want to know at other points
  - these points are called the data → “approximation of data”
- **Interpolation:**
  - find a function that goes exactly through data point
- **Regression:**
  - find a function that minimizes some error measure
  - better for noisy data.
- Related terms: curve fitting, extrapolation, classification

# Interpolation

- In the study of Geysers, an important quantity is the internal energy of steam.

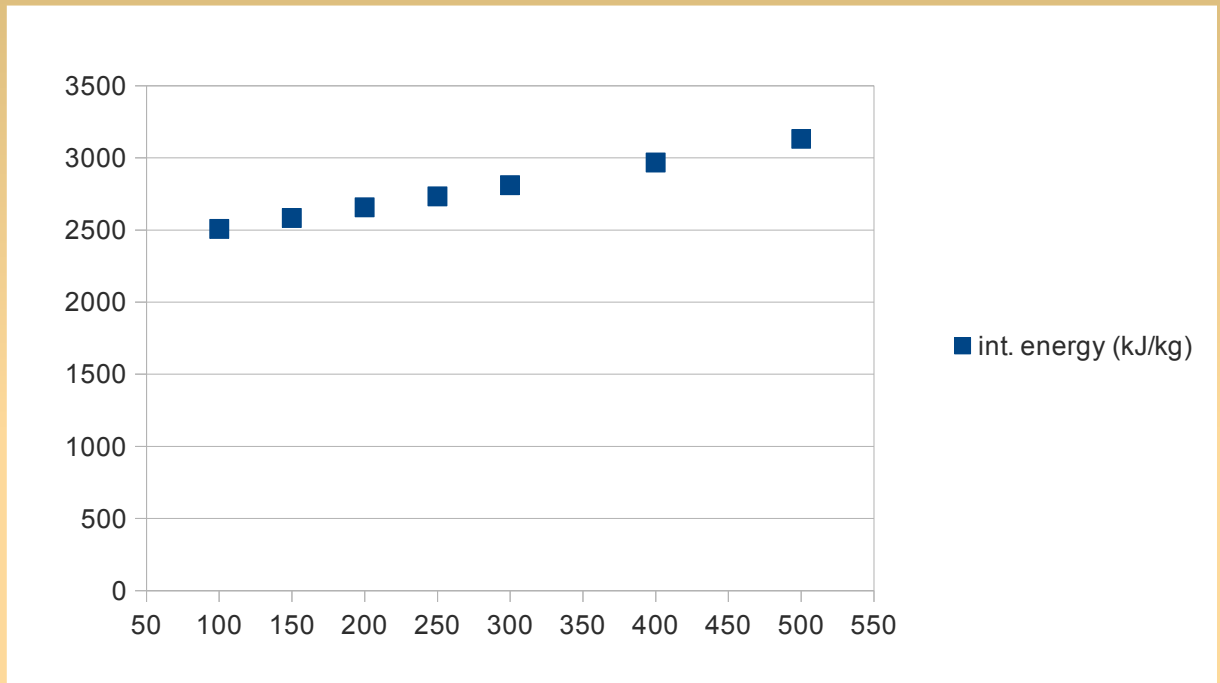
Temp. (Celsius)	int. energy (kJ/kg)
100	2506.7
150	2582.8
200	2658.1
250	2733.7
300	2810.4
400	2967.9
500	3131.6



(from Etter, 2011, Introduction to MATLAB)

# Temperature Example

Temp. (Celsius)	int. energy (kJ/kg)
100	2506.7
150	2582.8
200	2658.1
250	2733.7
300	2810.4
400	2967.9
500	3131.6

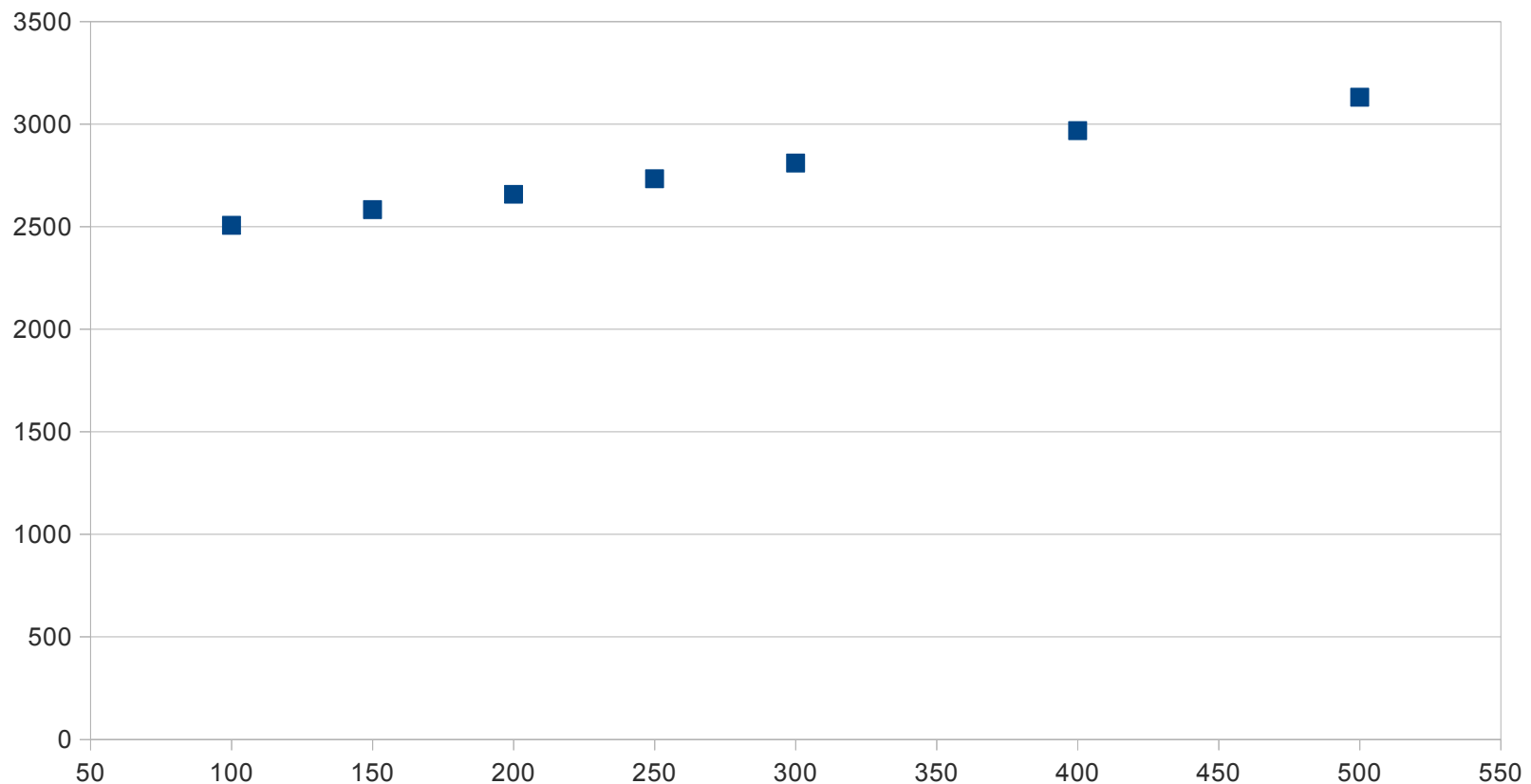


- Now we want to know the temp. at 430°C...



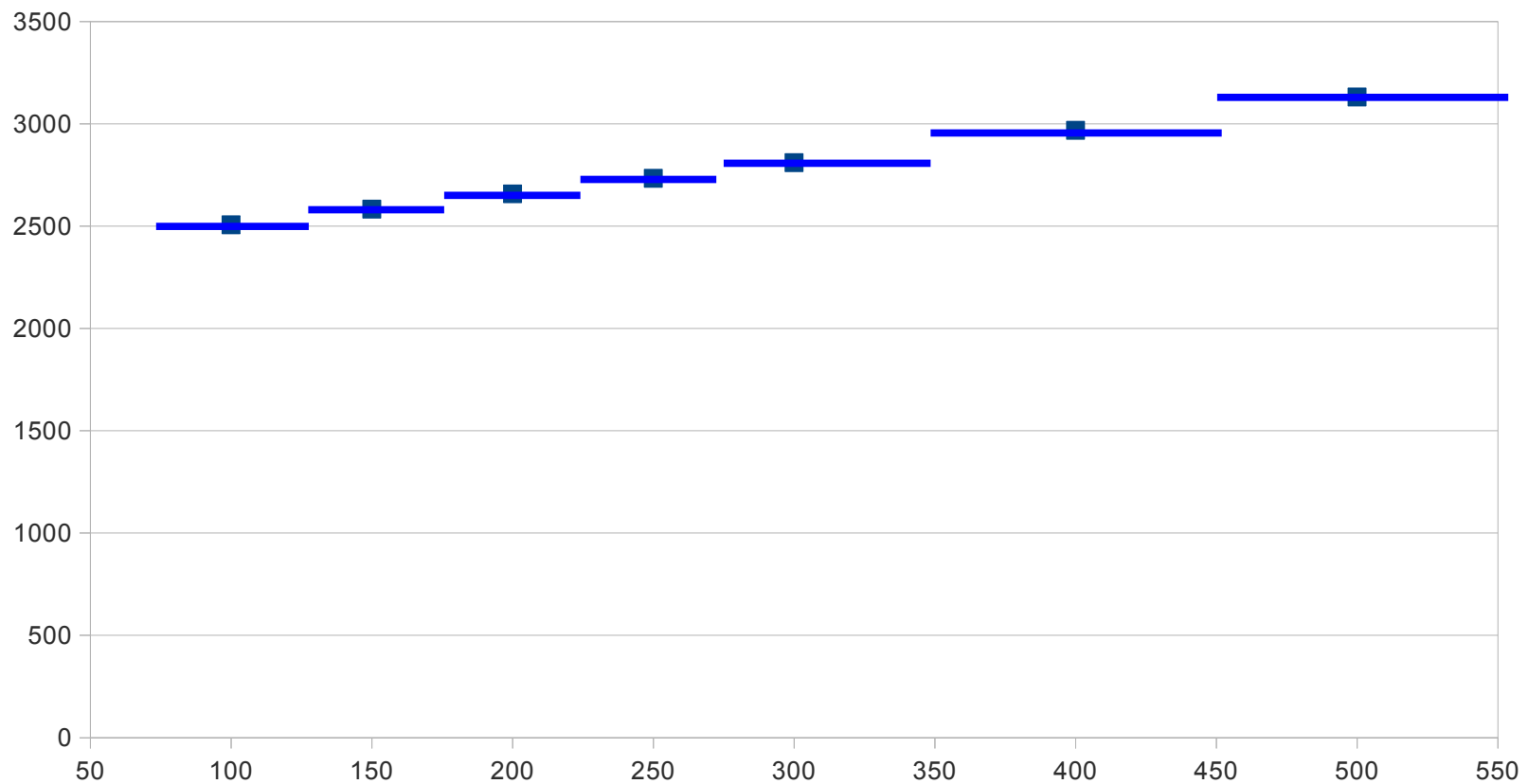
# Piecewise Constant Interpolation

- Interpolation: define a function that goes through data
- Piecewise interpolation: use a piecewise function



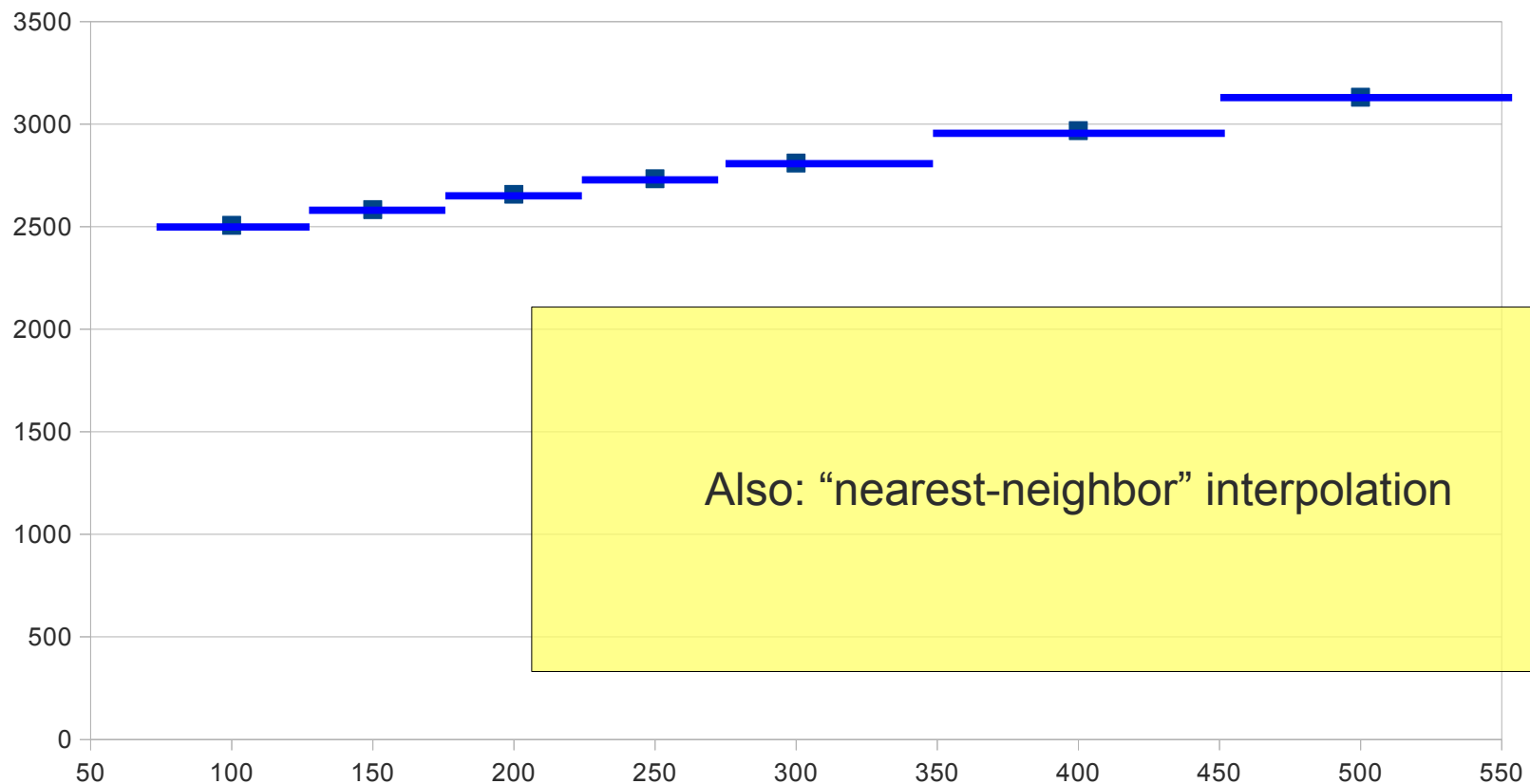
# Piecewise Constant Interpolation

- Interpolation: define a function that goes through data
- Piecewise interpolation: use a piecewise function



# Piecewise Constant Interpolation

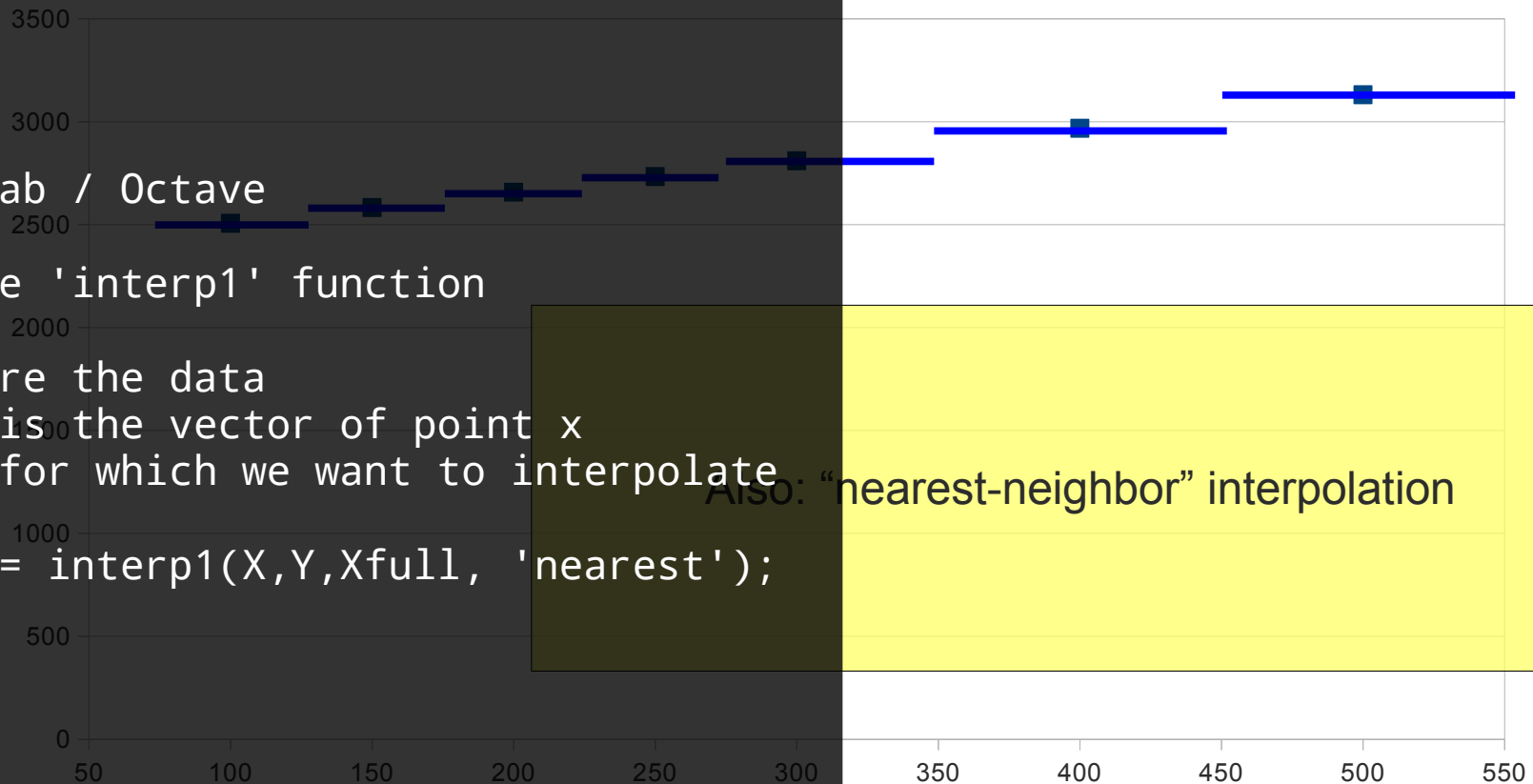
- Interpolation: define a function that goes through data
- Piecewise interpolation: use a piecewise function



# Piecewise Constant Interpolation

- Interpolation: define a function that goes through data
- Piecewise interpolation: use a piecewise function

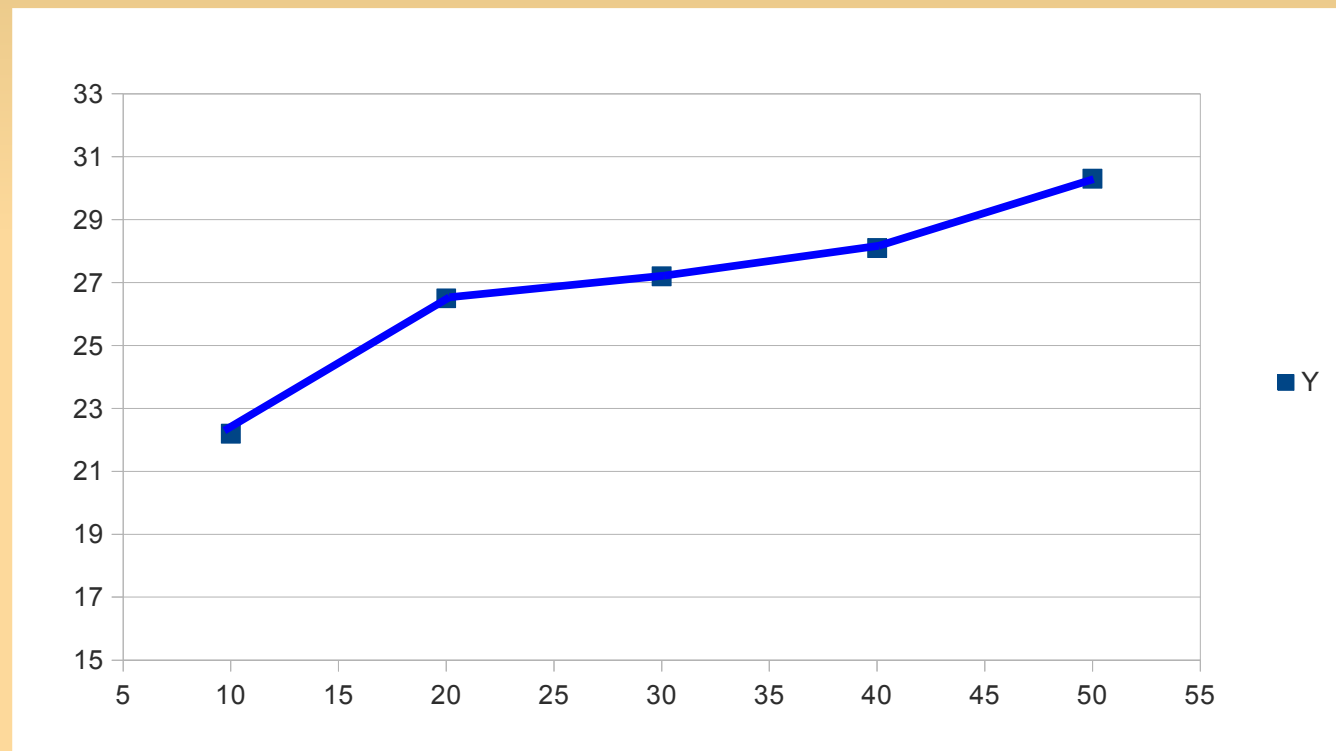
```
%In Matlab / Octave
%
% use the 'interp1' function
%
% X, Y are the data
% Xfull is the vector of point x
%   for which we want to interpolate
Yfull_n = interp1(X,Y,Xfull, 'nearest');
```



# Piecewise Linear Interpolation

- Piecewise linear interpolation:  
just connect the data point with lines

X	Y
10	22.2
20	26.5
30	27.2
40	28.1
50	30.3



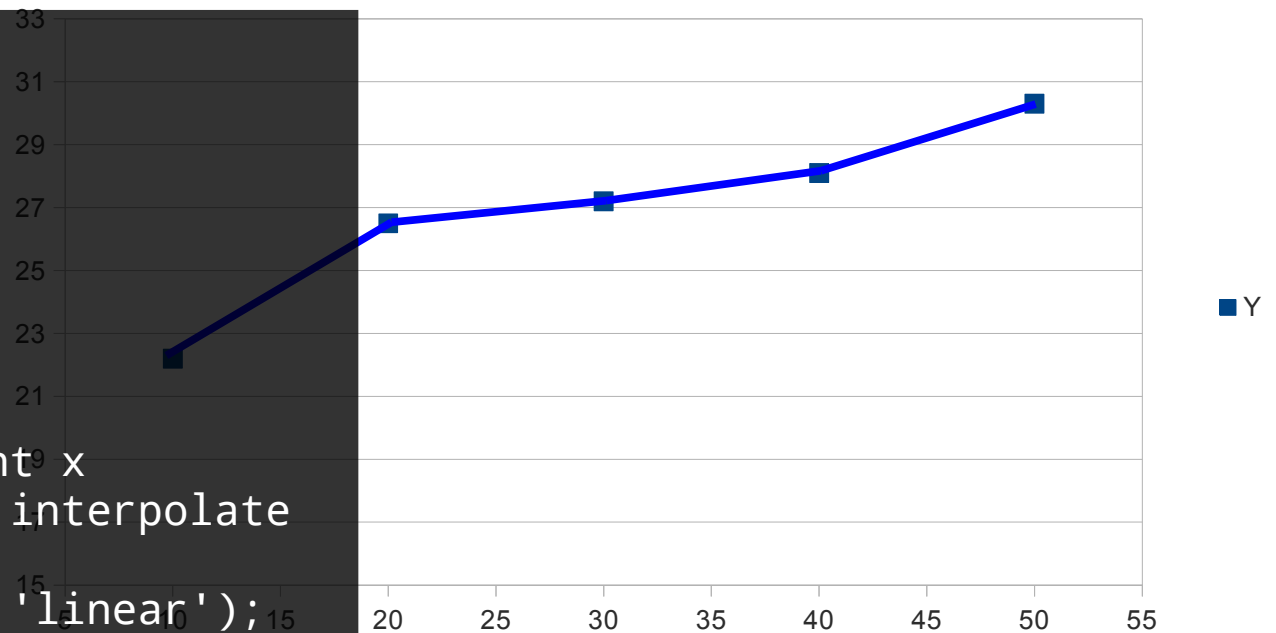
# Piecewise Linear Interpolation

- Piecewise linear interpolation:  
just connect the data point with lines

```
X      Y
    10  22.2
    20  26.5
    30  27.2
    40  28.1
    50  30.3

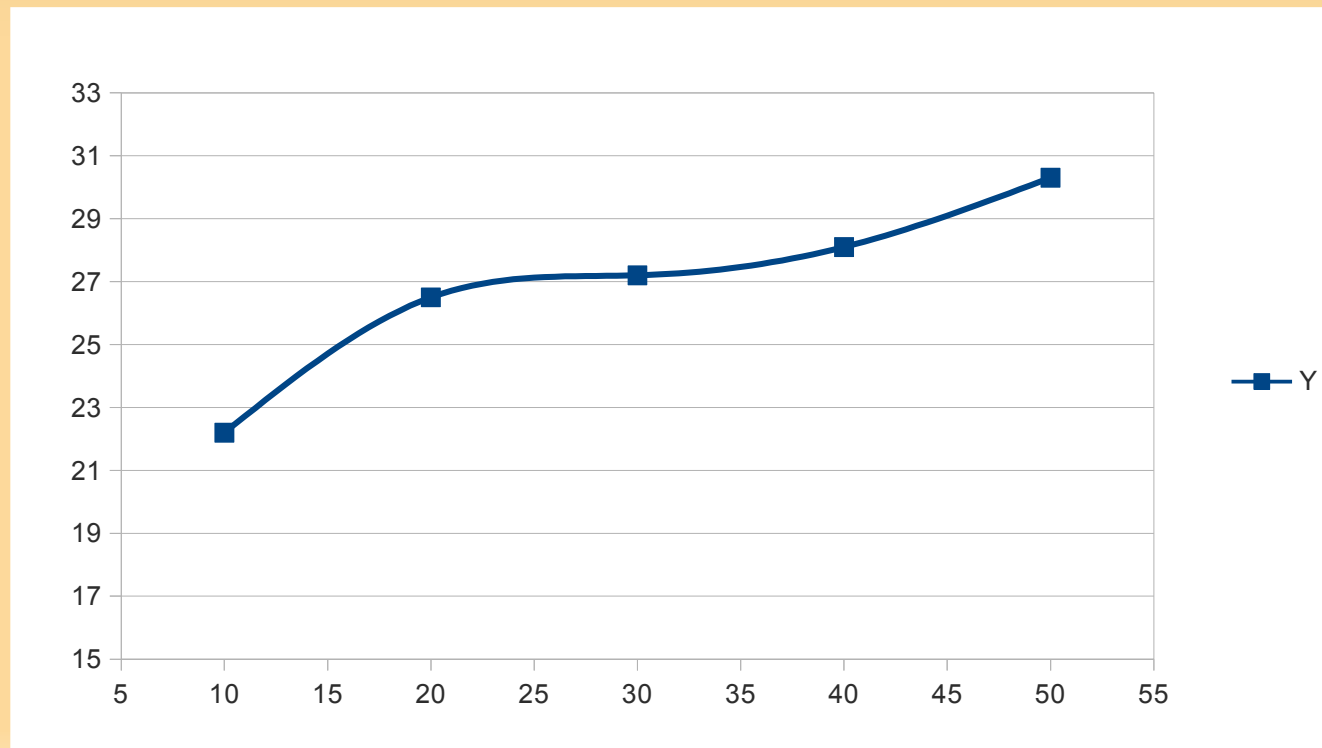
%In Matlab / Octave
%
% use the 'interp1' function
%
% X, Y are the data
% Xfull is the vector of point x
%   for which we want to interpolate

Yfull_n = interp1(X,Y,Xfull, 'linear');
```



# Cubic Splines Interpolation

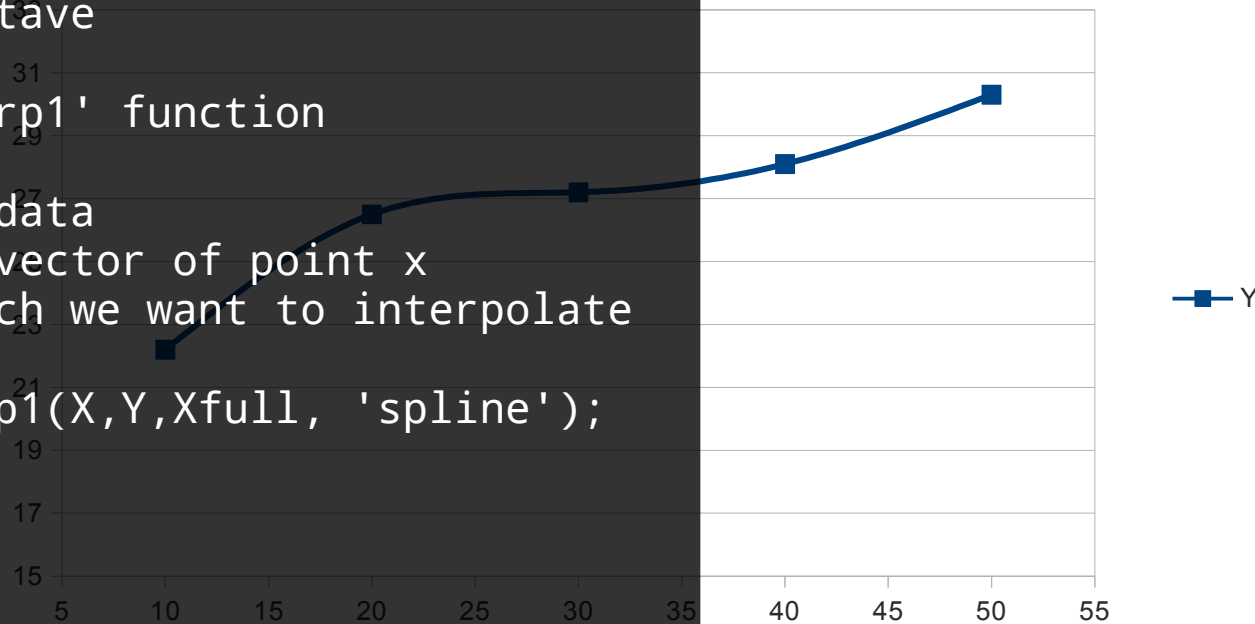
- **cubic-spline** interpolation
  - connect the data point smooth curves (third degree polynomials)
  - still piecewise



# Cubic Splines Interpolation

- **cubic-spline** interpolation
  - connect the data point smooth curves (third degree polynomials)
  - still piecewise

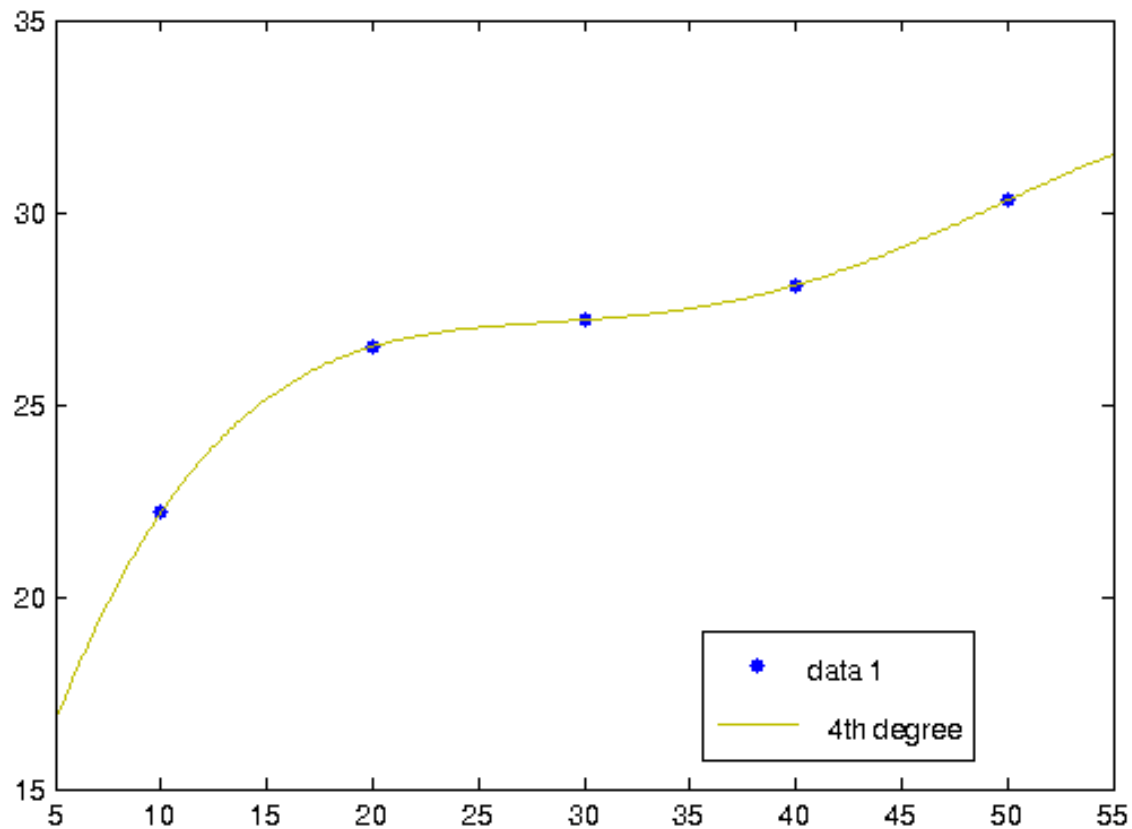
```
%In Matlab / Octave
%
% use the 'interp1' function
%
% X, Y are the data
% Xfull is the vector of point x
%     for which we want to interpolate
Yfull_n = interp1(X,Y,Xfull, 'spline');
```





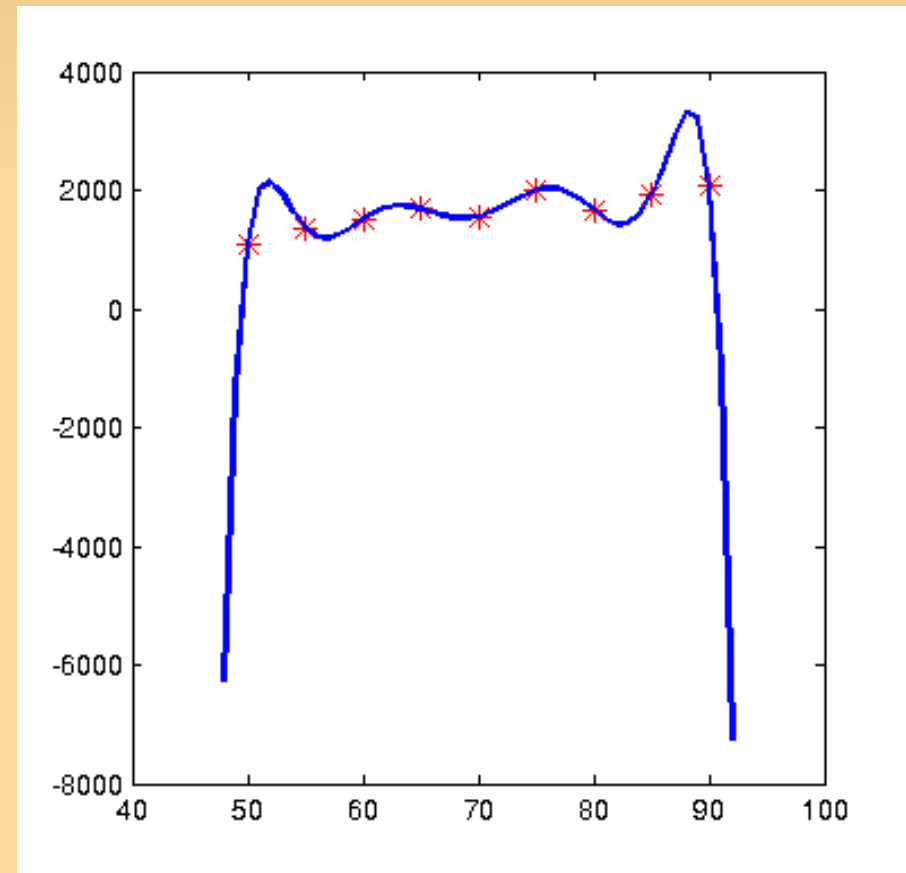
# Polynomial Interpolation

- So far: piecewise
- but may want to find a single (non-piecewise) function.



# Limits of Polynomial Interpolation

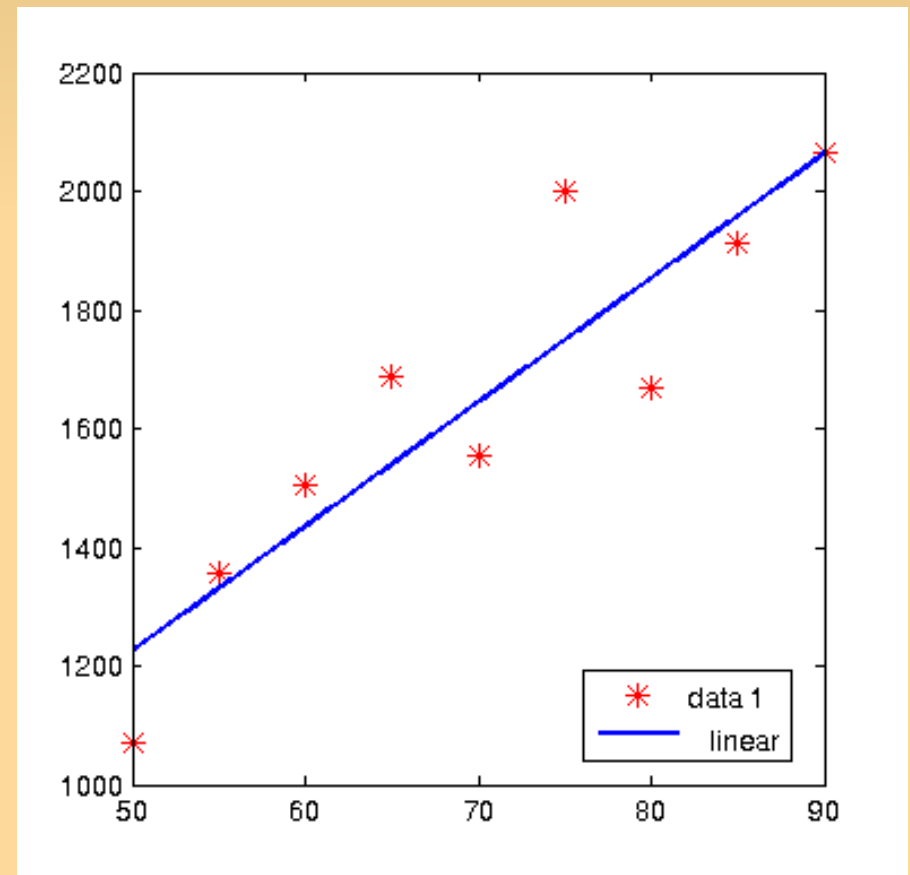
- Does not work very well when  $N$  is large.
- Is not very suitable if the data is obtained from noisy measurements.
- “Runge's phenomenon”
- In this case, we would perhaps want to fit a straight line.



# Least-Squares Method

- In cases that we made noisy measurements, we don't want to exactly fit the data.
- That is: fit a polynomial\*\* of degree  $p < n$ 
  - can still use 'polyfit'

\*\* or other function



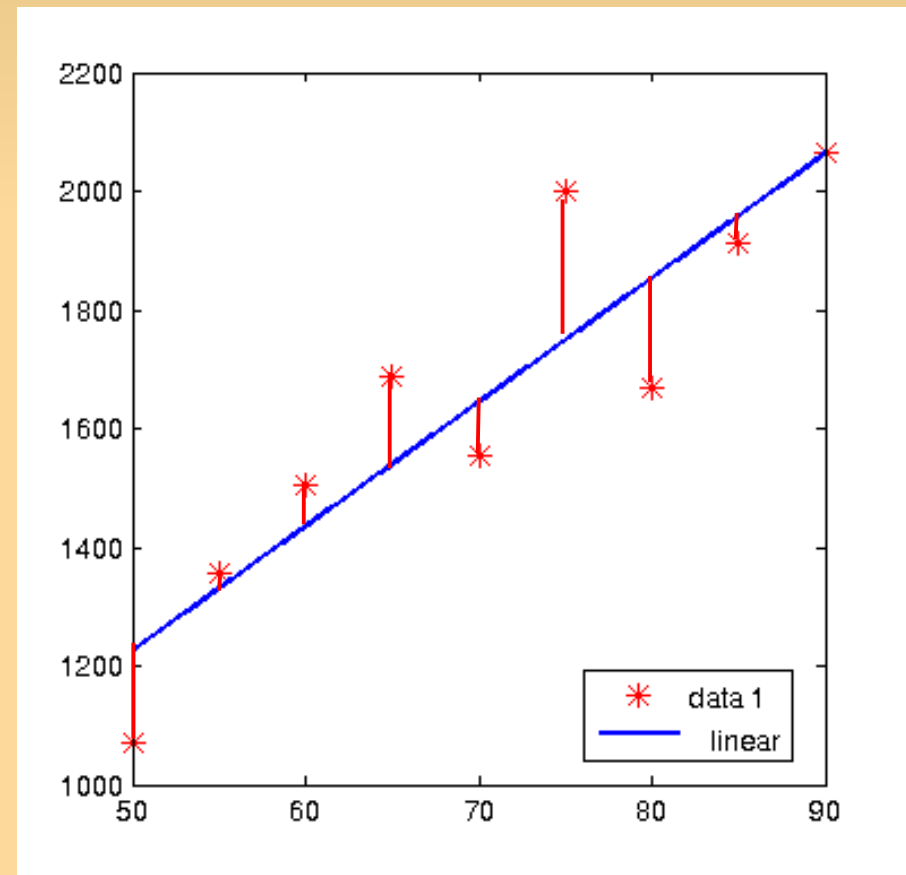
# Least-Squares Method

- Common approach:  
minimize sum of the squares of the errors

$$\tilde{f}(x) = a_0 + a_1 x$$

$$SSE(\tilde{f}) = \sum_{i=0}^n [\tilde{f}(x_i) - y_i]^2$$

- pick the  $\tilde{f}$  with min. SSE



Extra / old slides

# Polynomial Interpolation

- Polynomial interpolation: fit a polynomial

(Prop. 3.1)

given a set of  $N = n + 1$  data points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$

→ There exist a unique polynomial

$$\Pi_n(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$$

(of degree  $n$  or less) that goes exactly through the points!

- “The interpolating polynomial” (of the 'data' or 'function')

# Polynomial Interpolation

- Polynomial interpolation: fit a polynomial

(Prop. 3.1)

given a set of  $N = n + 1$  data points  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$

→ There exist a unique polynomial

$$\Pi_n(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$$

(of degree  $n$  or less) that goes exactly through the points!

- “The interpolating polynomial” (of the 'data' or 'function')

So this is good news -  
we can always find such a function.

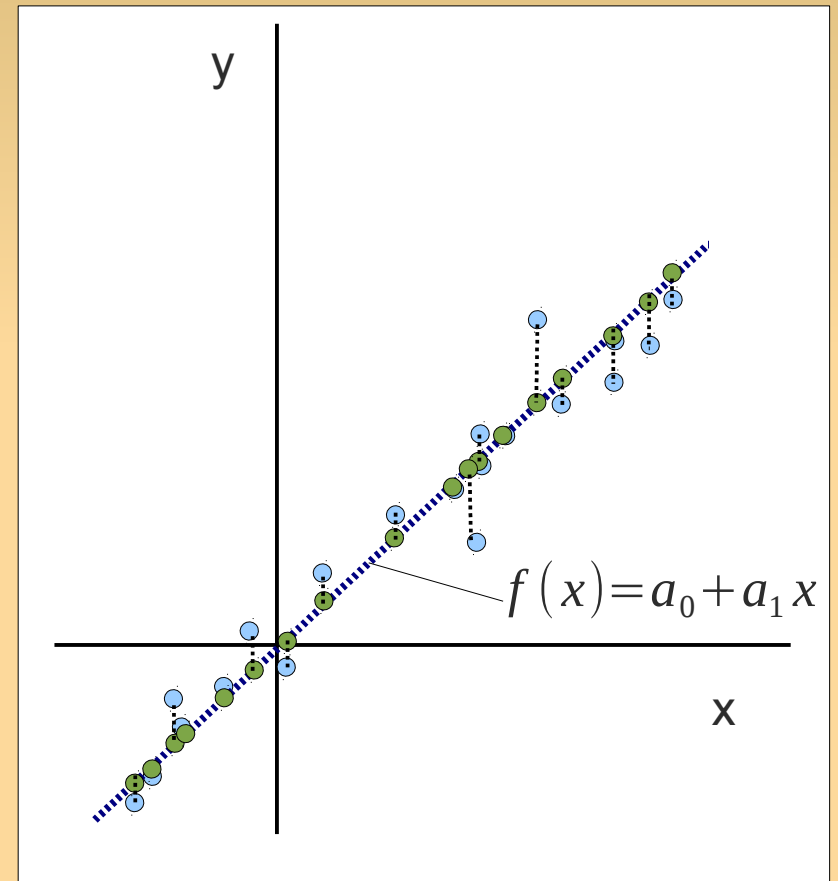
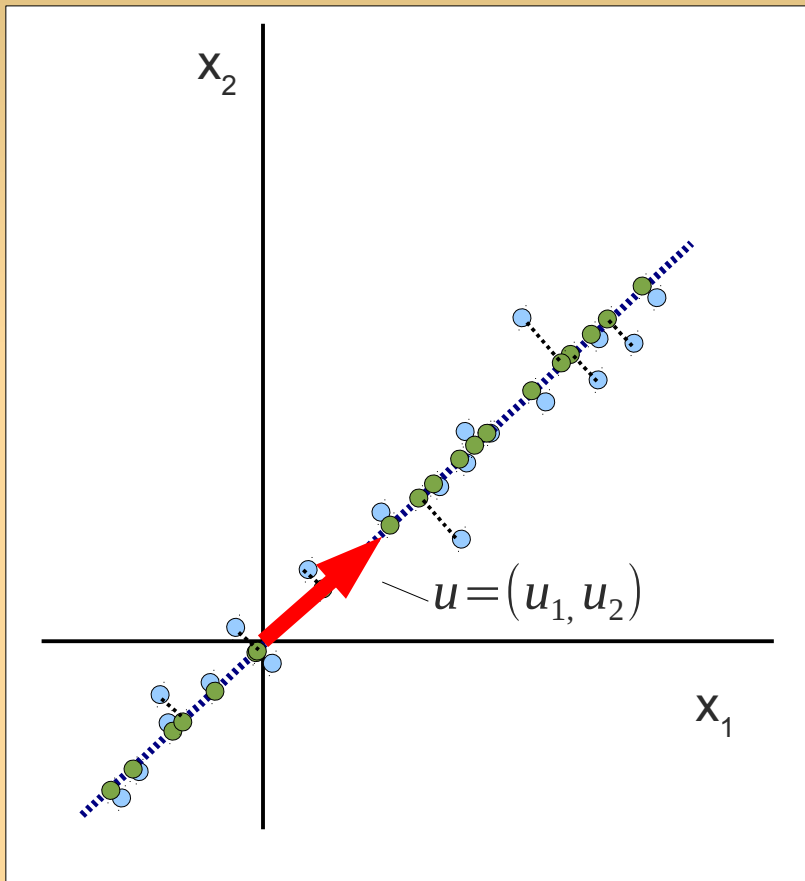
# Uniqueness of the Interpolating polynomial

- Why is this polynomial unique?
- Suppose not unique: both  $\Pi_n(x), \Pi_n'(x)$  perfectly fit the data
  - $\Pi_n(x) - \Pi_n'(x) = 0$  for all the  $N=n+1$  data points
- That is it
  - “vanishes at  $n+1$  points”
  - “has  $n+1$  roots”
- But: a polynomial of degree  $n$  has at most  $n$  roots!
  - contradiction!



# PCA vs. Least Squares

- What would happen when switching the axes...?



# PCA vs. Least Squares

- What would happen when switching the axes...?

