Applied Machine Learning

Linear Regression

Siamak Ravanbakhsh

Learning objectives

- linear model
- evaluation criteria
- how to find the best fit
- geometric interpretation

Motivation

History: method of least squares was invented by **Legendre** and **Gauss** (1800's)

Gauss at age 24 used it to predict the future location of Ceres (largest astroid in the astroid belt)

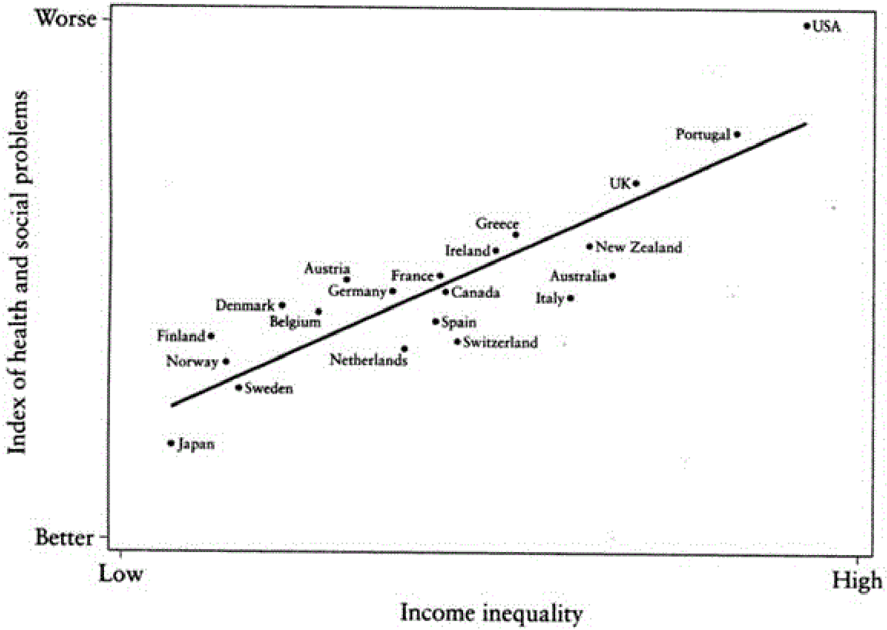
source: http://chrisauld.com/2012/10/07/what-do-we-know-about-the-effect-of-income-inequality-on-health/

Motivation

History: method of least squares was invented by **Legendre** and **Gauss** (1800's)

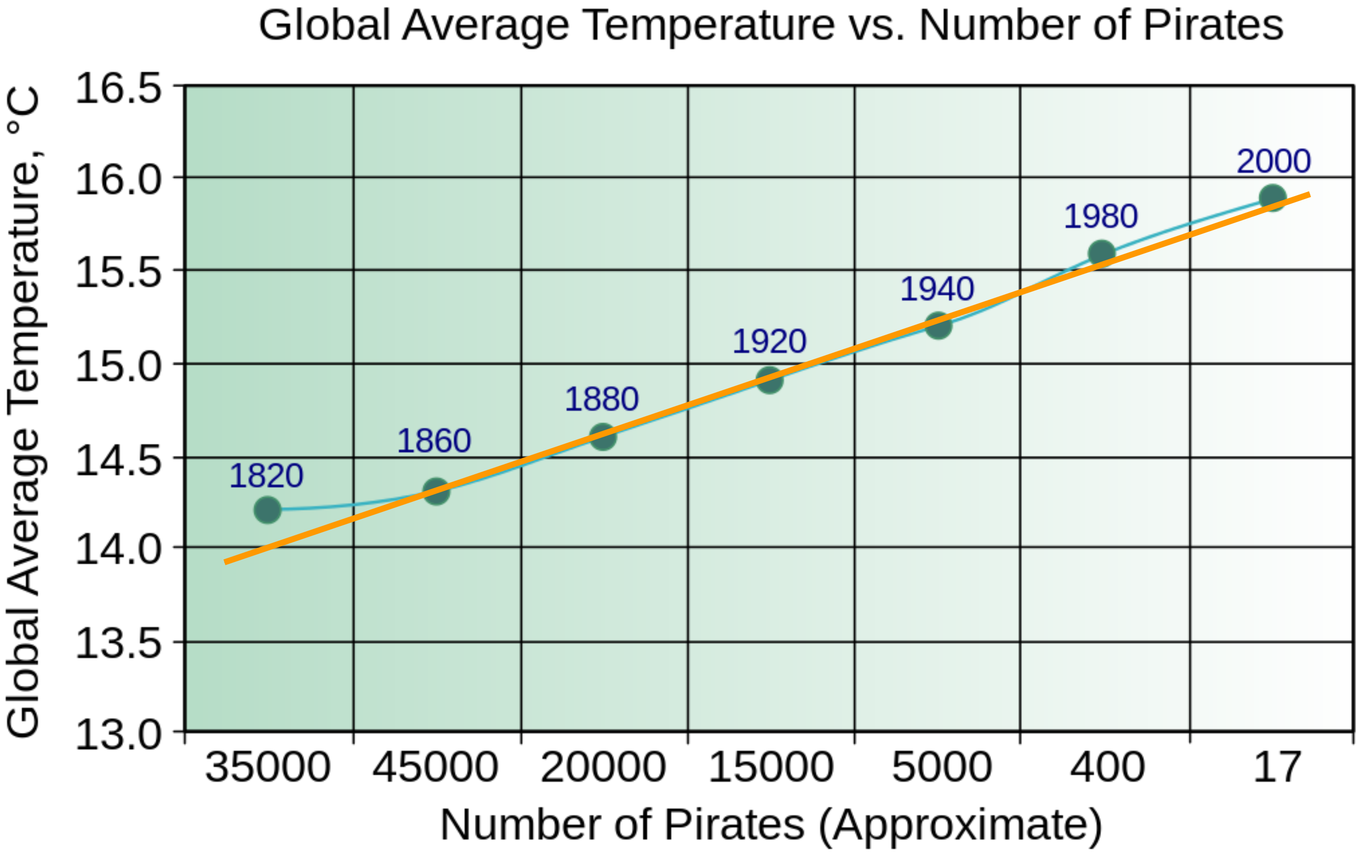
Gauss at age 24 used it to predict the future location of Ceres (largest astroid in the astroid belt)

effect of income inequality on health and social problems



source: http://chrisauld.com/2012/10/07/what-do-we-know-about-the-effect-of-income-inequality-on-health/

Motivation (?)


Global Average Temperature vs. Number of Pirates
Global Average Temperature, °C
16.5
16.0
15.5
15.0
14.5
14.0
13.5
13.0
1820
1860
1880
1920
1940
1980
2000
35000
45000
20000
15000
5000
400
17
Number of Pirates (Approximate)

Representing data

each instance: $\left| \begin{array}{l} x^{(n)} \in \mathbb{R}^D \\ y^{(n)} \in \mathbb{R} \end{array} \right.$

Representing data

one instance

each instance: $\left| \begin{array}{l} x^{(n)} \in \mathbb{R}^D \\ y^{(n)} \in \mathbb{R} \end{array} \right.$

Representing data

one instance

each instance: $\left| \begin{array}{l} x^{(n)} \in \mathbb{R}^D \\ y^{(n)} \in \mathbb{R} \end{array} \right.$

vectors are assume to be column vectors $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} x_1, & x_2, & \dots, & x_D \end{bmatrix}^\top$

Representing data

one instance

each instance: $\left| \begin{array}{l} x^{(n)} \in \mathbb{R}^D \\ y^{(n)} \in \mathbb{R} \end{array} \right.$

a feature

vectors are assume to be column vectors $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} x_1, & x_2, & \ldots, & x_D \end{bmatrix}^\top$

Representing data

one instance

each instance: $\left| \begin{array}{l} x^{(n)} \in \mathbb{R}^D \\ y^{(n)} \in \mathbb{R} \end{array} \right.$

a feature

vectors are assume to be column vectors $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} x_1, & x_2, & \ldots, & x_D \end{bmatrix}^\top$

we assume N instances in the dataset $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^N$

each instance has D features indexed by d

for example, $x_d^{(n)} \in \mathbb{R}$ is the feature d of instance n

Representing data

design matrix: *concatenate all instances*

- *each row is a datapoint, each column is a feature*

$$X = \begin{bmatrix} {x^{(1)}}^\top \\ {x^{(2)}}^\top \\ \vdots \\ {x^{(N)}}^\top \end{bmatrix}$$

Representing data

design matrix: *concatenate all instances*

- *each row is a datapoint, each column is a feature*

$$X = \begin{bmatrix} {x^{(1)}}^\top \\ {x^{(2)}}^\top \\ \vdots \\ {x^{(N)}}^\top \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix}$$

Representing data

design matrix: *concatenate all instances*

- *each row is a datapoint, each column is a feature*

$$X = \begin{bmatrix} {x^{(1)}}^\top \\ {x^{(2)}}^\top \\ \vdots \\ {x^{(N)}}^\top \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix}$$

one instance

Representing data

design matrix: *concatenate all instances*

- *each row is a datapoint, each column is a feature*

$$X = \begin{bmatrix} {x^{(1)}}^\top \\ {x^{(2)}}^\top \\ \vdots \\ {x^{(N)}}^\top \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix}$$

one instance

one feature

Representing data

design matrix: *concatenate all instances*

- *each row is a datapoint, each column is a feature*

$$X = \begin{bmatrix} {x^{(1)}}^{\top} \\ {x^{(2)}}^{\top} \\ \vdots \\ {x^{(N)}}^{\top} \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times D}$$

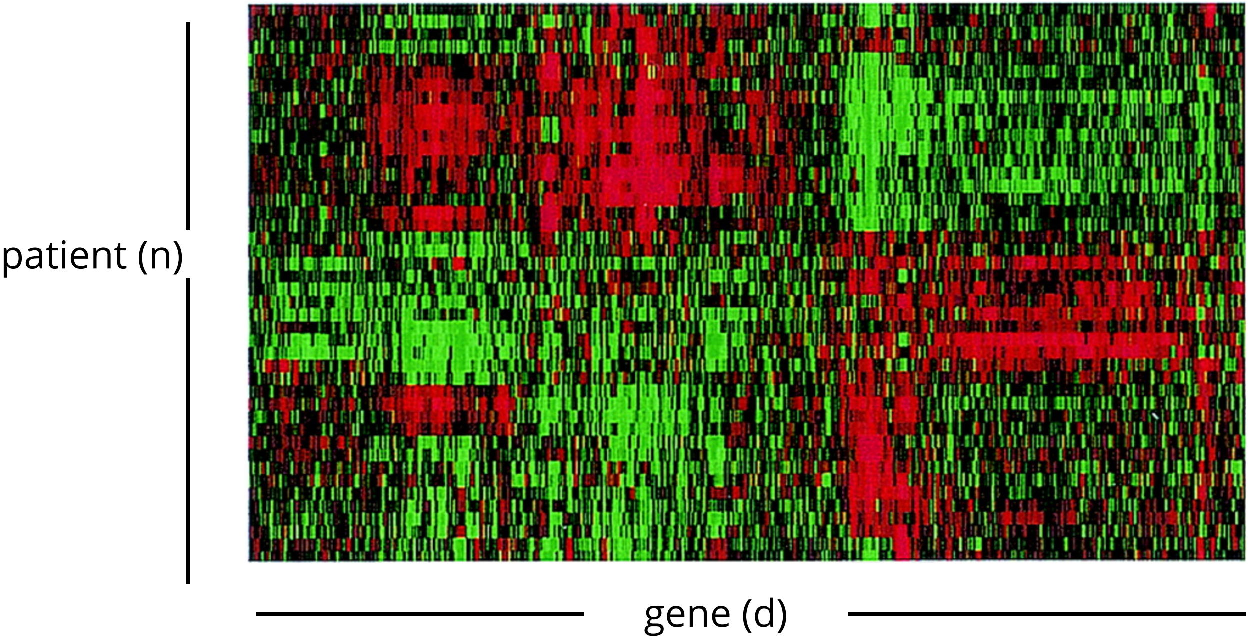
one instance

one feature

Representing data

Example:

Micro array data (X), contains gene expression levels
labels (y) can be {cancer/no cancer} label for each patient



$$\in \mathbb{R}^{N \times D}$$

Linear model

assuming a scalar output $f_w : \mathbb{R}^D \to \mathbb{R}$

will generalize to a vector later

$$f_w(x) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

Linear model

assuming a scalar output $f_w : \mathbb{R}^D \rightarrow \mathbb{R}$

will generalize to a vector later

$$f_{w}(x) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

↓ model parameters or weights

Linear model

assuming a scalar output $f_w : \mathbb{R}^D \to \mathbb{R}$

will generalize to a vector later

$$f_w(x) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

model parameters or weights

bias or intercept

Linear model

assuming a scalar output $f_w : \mathbb{R}^D \to \mathbb{R}$

will generalize to a vector later

$$f_w(x) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

model parameters or weights

bias or intercept

simplification

concatenate a 1 to x $\longrightarrow$ $x = [1, x_1, \ldots, x_D]^\top$

$$f_w(x) = w^\top x$$

Linear model

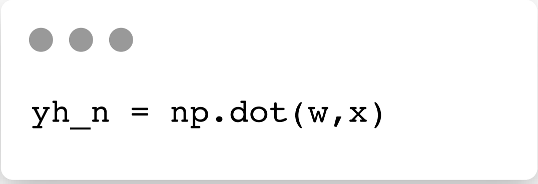
assuming a scalar output $f_w : \mathbb{R}^D \to \mathbb{R}$

will generalize to a vector later

$$f_w(x) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

model parameters or weights

bias or intercept



```
yh_n = np.dot(w,x)
```

simplification

concatenate a 1 to x $\longrightarrow$ $x = [1, x_1, \ldots, x_D]^\top$

$$f_w(x) = w^\top x$$

Loss function

objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

Loss function

objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

$$f_w(x^{(n)}) \approx y^{(n)}$$

Loss function

objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

$$f_w(x^{(n)}) \approx y^{(n)}$$

minimize a measure of difference between $\hat{y}^{(n)} = f_w(x^{(n)})$ and $y^{(n)}$

Loss function

objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

$$f_w(x^{(n)}) \approx y^{(n)}$$

minimize a measure of difference between $\hat{y}^{(n)} = f_w(x^{(n)})$ and $y^{(n)}$

square error **loss** (a.k.a. **L2** loss) $L(y, \hat{y}) \triangleq \frac{1}{2}(y - \hat{y})^2$

for a single instance (a function of labels)

Loss function

objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

$$f_w(x^{(n)}) \approx y^{(n)}$$

minimize a measure of difference between $\hat{y}^{(n)} = f_w(x^{(n)})$ and $y^{(n)}$

square error **loss** (a.k.a. **L2** loss) $L(y, \hat{y}) \triangleq \frac{1}{2}(y - \hat{y})^2$

for a single instance (a function of labels)

for future convenience

Loss function

objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

$$\boxed{f_w(x^{(n)}) \approx y^{(n)}}$$

minimize a measure of difference between $\hat{y}^{(n)} = f_w(x^{(n)})$ and $y^{(n)}$

square error **loss** (a.k.a. **L2** loss) $L(y, \hat{y}) \triangleq \frac{1}{2}(y - \hat{y})^2$

for a single instance (a function of labels)

for future convenience

sum of squared errors **cost function**

$$J(w) = \frac{1}{2} \sum_{n=1}^{N} \left(y^{(n)} - w^\top x^{(n)}\right)^2$$

Loss function

objective: find parameters to fit the data $x^{(n)}, y^{(n)} \quad \forall n$

$$f_w(x^{(n)}) \approx y^{(n)}$$

minimize a measure of difference between $\hat{y}^{(n)} = f_w(x^{(n)})$ and $y^{(n)}$

square error **loss** (a.k.a. **L2** loss) $L(y, \hat{y}) \triangleq \frac{1}{2}(y - \hat{y})^2$

for a single instance (a function of labels)

for future convenience

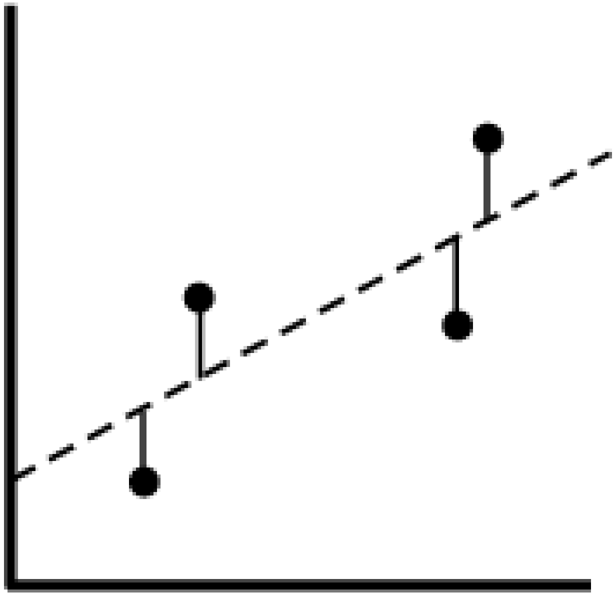
versus

for the whole dataset

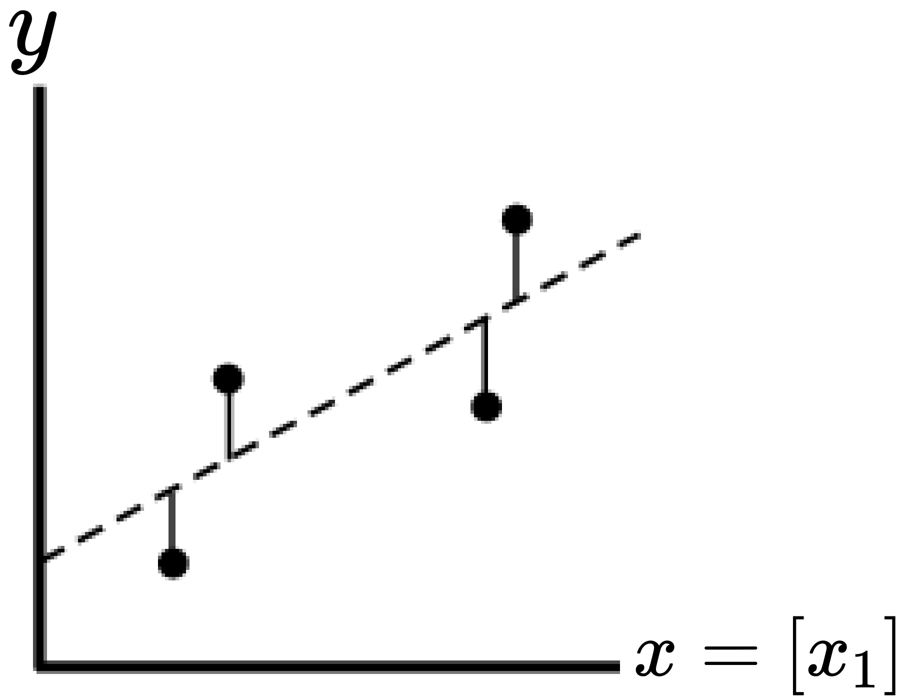
sum of squared errors **cost function**

$$J(w) = \frac{1}{2} \sum_{n=1}^{N} \left(y^{(n)} - w^\top x^{(n)} \right)^2$$

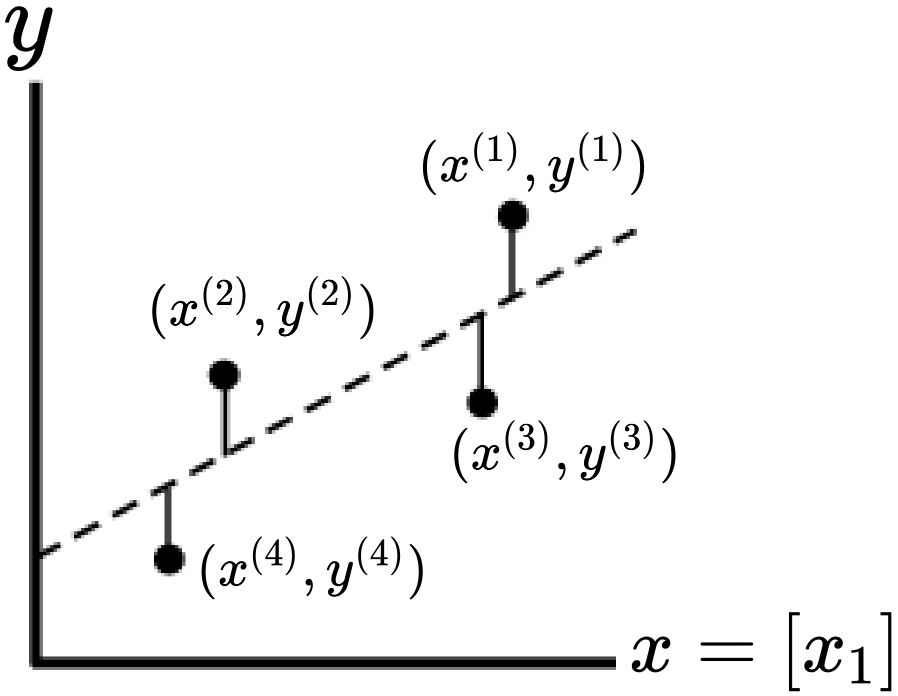
Example (D = 1) +bias (D=2)!



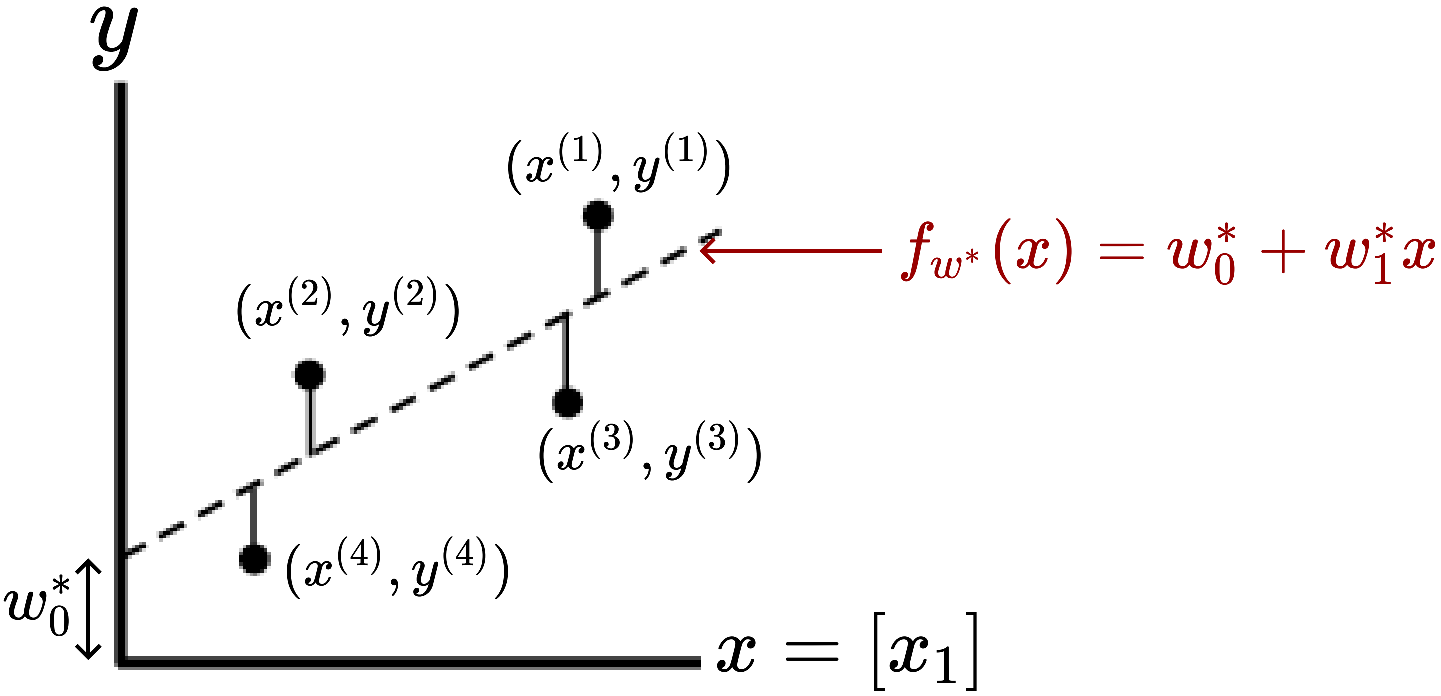
Example (D = 1) +bias (D=2)!



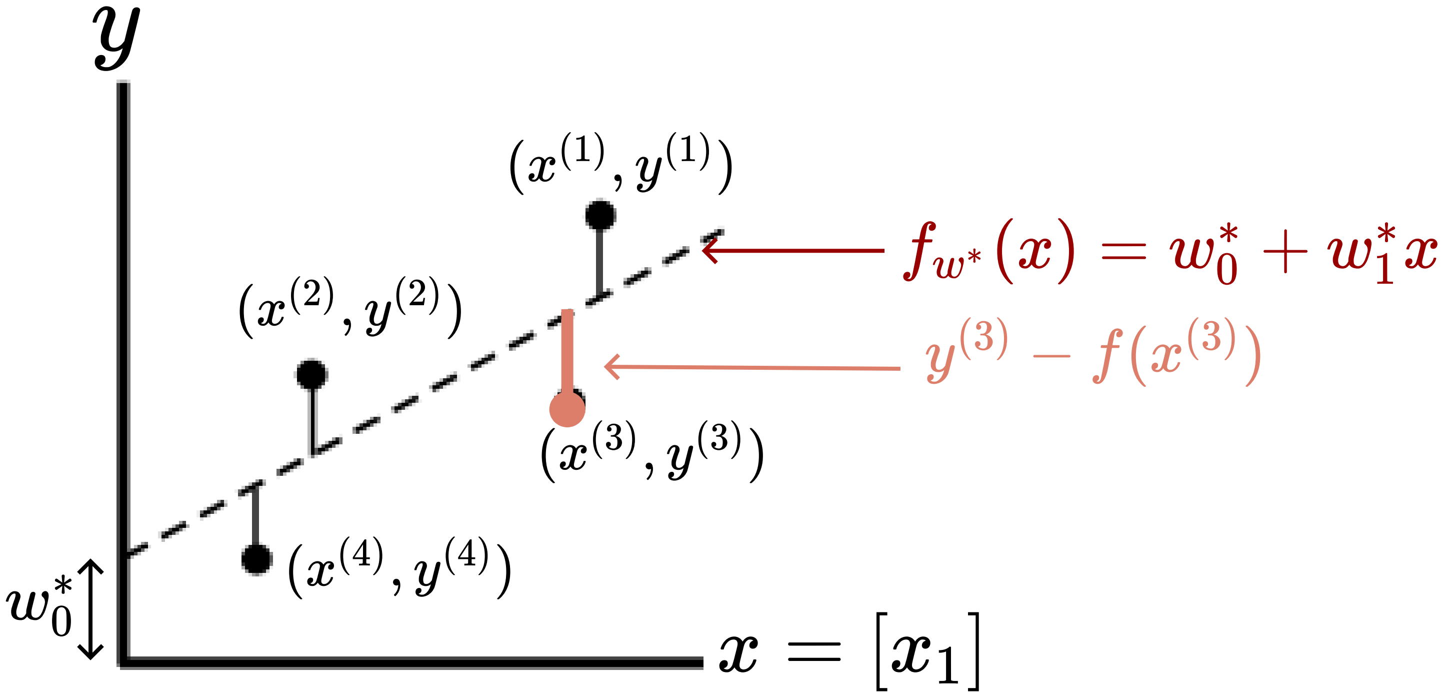
Example (D = 1) +bias (D=2)!



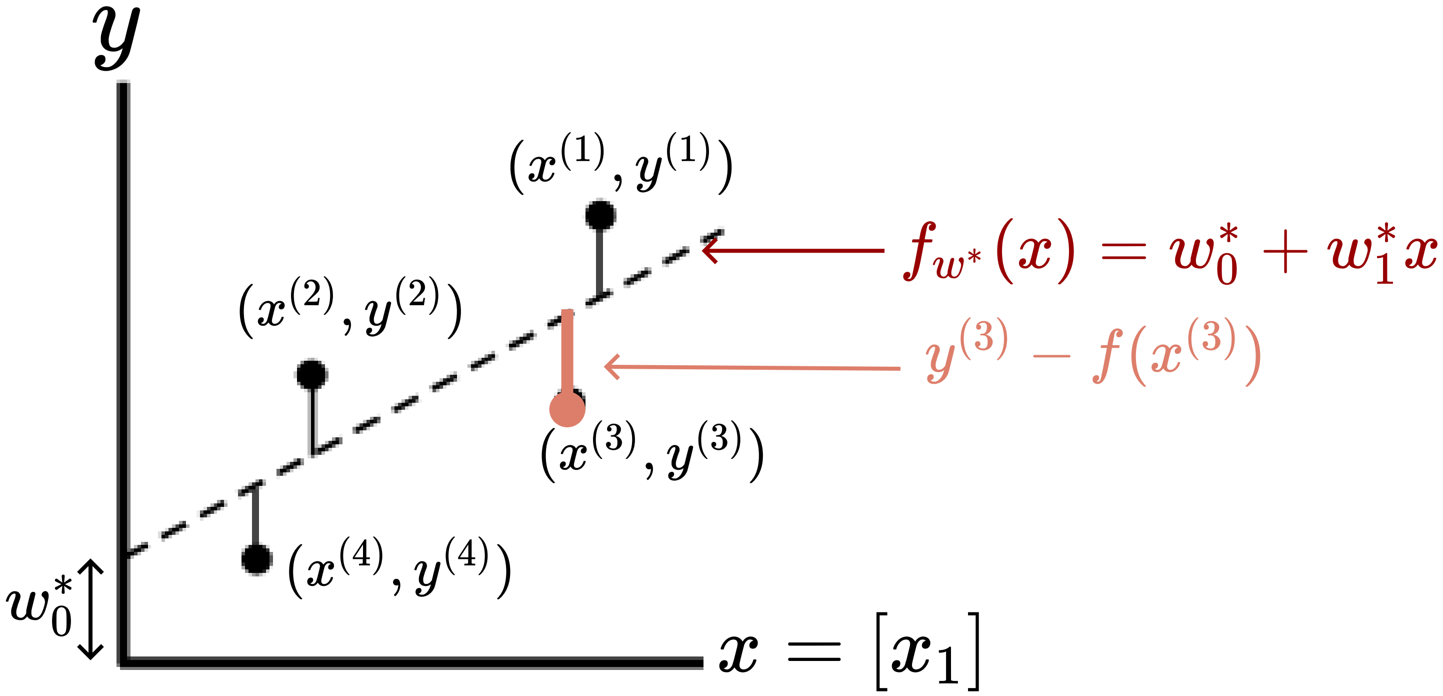
Example (D = 1) +bias (D=2)!



Example (D = 1) +bias (D=2)!

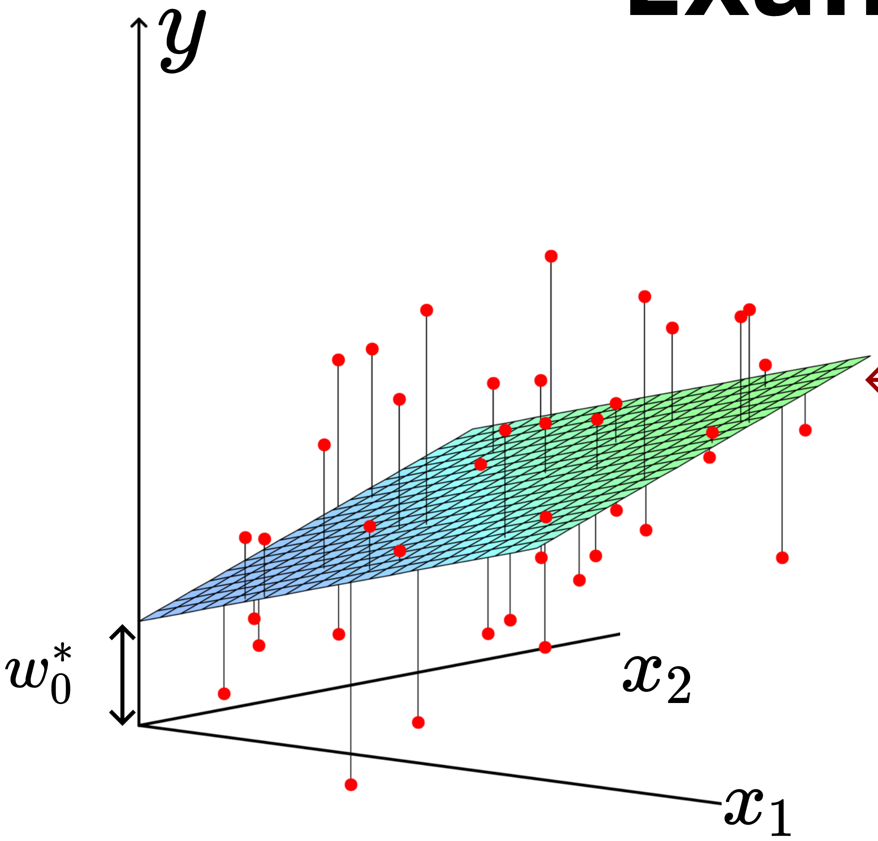


Example (D = 1) +bias (D=2)!



Linear Least Squares $\min_w \sum_n \left(y^{(n)} - w^T x^{(n)} \right)^2$

Example (D=2) +bias (D=3)!



$$f_{w^*}(x) = w_0^* + w_1^* x_1 + w_2^* x_2$$

Linear Least Squares $w^* = \arg\min_w \sum_n \left(y^{(n)} - w^T x^{(n)}\right)^2$

Matrix form

instead of $\underset{\in \mathbb{R}}{\hat{y}^{(n)}} = \underset{1 \times D}{w}^{\top} \underset{D \times 1}{x^{(n)}}$

Matrix form

instead of $\underset{\in \mathbb{R}}{\hat{y}^{(n)}} = \underset{1 \times D}{w^\top} \underset{D \times 1}{x^{(n)}}$

use **design matrix** to write $$\underset{N \times 1}{\hat{y}} = \underset{N \times D}{X} \underset{D \times 1}{w}$$

Matrix form

instead of $\underset{\in \mathbb{R}}{\hat{y}^{(n)}} = \underset{1 \times D}{w^\top} \underset{D \times 1}{x^{(n)}}$

use **design matrix** to write $\underset{N \times 1}{\hat{y}} = \underset{N \times D}{X} \underset{D \times 1}{w}$

Linear least squares

$$\arg\min_w \frac{1}{2}||y - Xw||^2 = \frac{1}{2}(y - Xw)^\top (y - Xw)$$

squared L2 norm of the **residual** vector

Matrix form

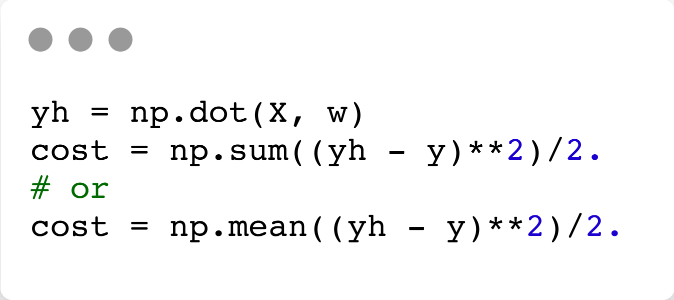
instead of $\underset{\in \mathbb{R}}{\hat{y}^{(n)}} = \underset{1 \times D}{w^\top} \underset{D \times 1}{x^{(n)}}$

use **design matrix** to write $\underset{N \times 1}{\hat{y}} = \underset{N \times D}{X} \underset{D \times 1}{w}$

Linear least squares

$$\arg\min_w \frac{1}{2}||y - Xw||^2 = \frac{1}{2}(y - Xw)^\top (y - Xw)$$

squared L2 norm of the **residual** vector



```
yh = np.dot(X, w)
cost = np.sum((yh - y)**2)/2.
# or
cost = np.mean((yh - y)**2)/2.
```

Minimizing the cost

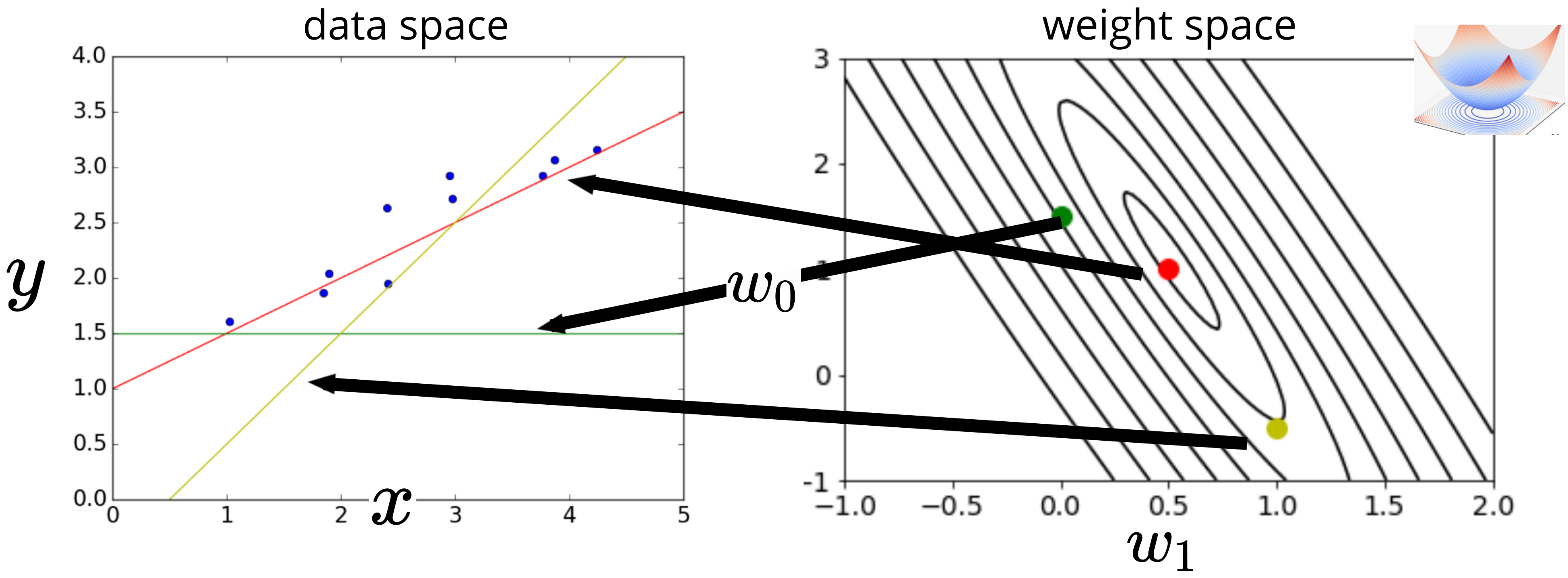
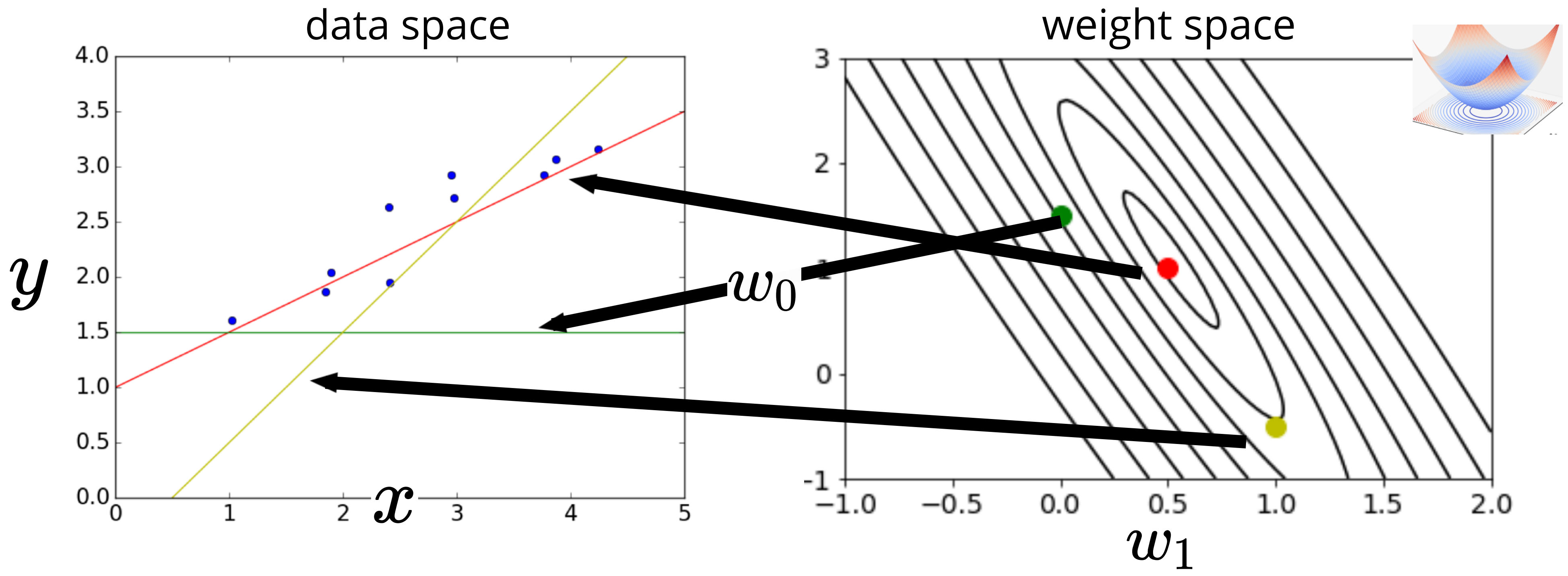


image: Grosse, Farahmand, Carrasquilla

Minimizing the cost



the objective is a smooth function of w

find minimum by setting partial derivatives to zero

image: Grosse, Farahmand, Carrasquilla

Simple case: D = 1

model $f_w(x) = \boxed{wx}$

both scalar

Simple case: D = 1

model $f_w(x) = \boxed{wx}$

both scalar

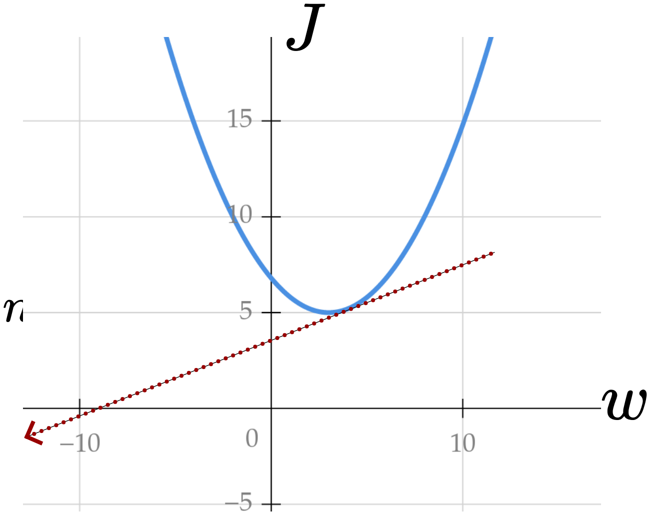
cost function $J(w) = \frac{1}{2}\sum_n (y^{(n)} - wx^{(n)})^2$

Simple case: D = 1

model $f_w(x) = wx$

both scalar

cost function $J(w) = \frac{1}{2}\sum_n (y^{(n)} - wx^{(n}$

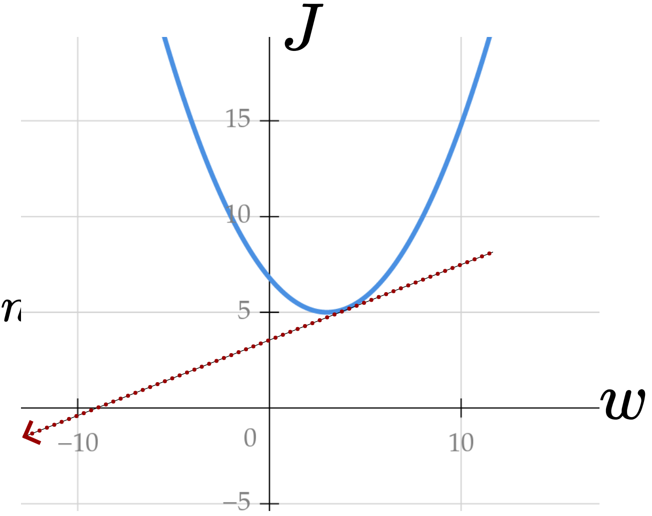


Simple case: D = 1

model $f_w(x) = \boxed{wx}$

both scalar

cost function $J(w) = \frac{1}{2}\sum_n (y^{(n)} - wx^{(n}$



derivative $\frac{\mathrm{d}J}{\mathrm{d}w} = \sum_n x^{(n)}(wx^{(n)} - y^{(n)})$

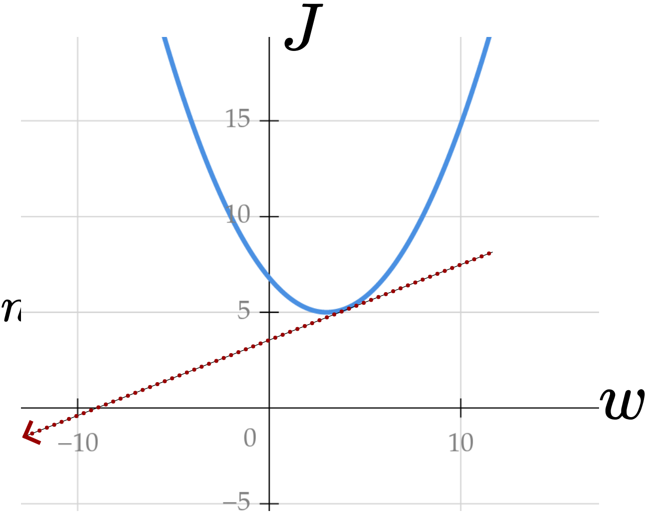
setting the derivative to zero $w = \frac{\sum_n x^{(n)} y^{(n)}}{\sum_n {x^{(n)}}^2}$

Simple case: D = 1

model $f_w(x) = \boxed{wx}$

both scalar

cost function $J(w) = \frac{1}{2}\sum_n (y^{(n)} - wx^{(n}$



derivative $\frac{\mathrm{d}J}{\mathrm{d}w} = \sum_n x^{(n)}(wx^{(n)} - y^{(n)})$

setting the derivative to zero $w = \frac{\sum_n x^{(n)} y^{(n)}}{\sum_n {x^{(n)}}^2}$

global minimum because cost is smooth and *convex*

more on convexity layer

Gradient

for a multivariate function $J(w_0, w_1)$

partial derivatives instead of derivative

= derivative when other vars. are fixed

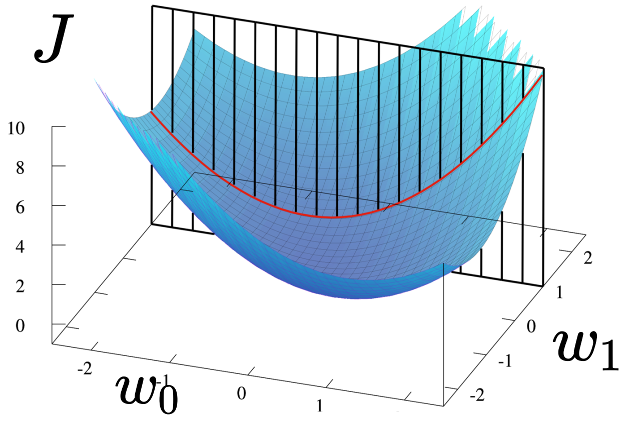
Gradient

for a multivariate function $J(w_0, w_1)$

partial derivatives instead of derivative

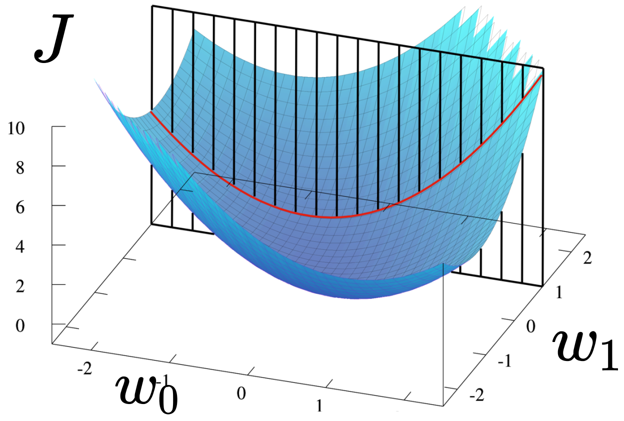
= derivative when other vars. are fixed

$$\frac{\partial}{\partial w_1} J(w_0, w_1) \triangleq \lim_{\epsilon \to 0} \frac{J(w_0, w_1+\epsilon) - J(w_0, w_1)}{\epsilon}$$



Gradient

for a multivariate function $J(w_0, w_1)$



partial derivatives instead of derivative

= derivative when other vars. are fixed

$$\frac{\partial}{\partial w_1} J(w_0, w_1) \triangleq \lim_{\epsilon \to 0} \frac{J(w_0, w_1+\epsilon) - J(w_0, w_1)}{\epsilon}$$

critical point: all partial derivatives are zero

Gradient

for a multivariate function $J(w_0, w_1)$

partial derivatives instead of derivative

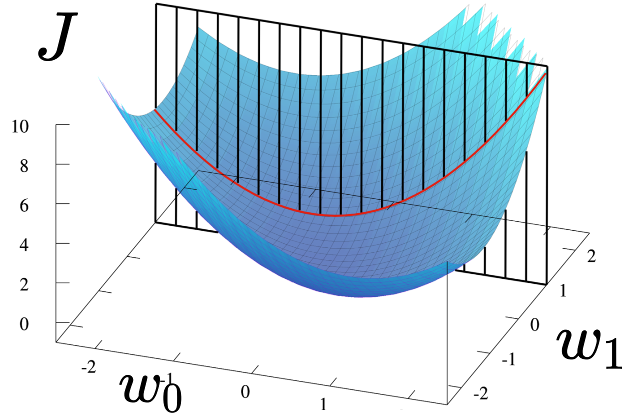
= derivative when other vars. are fixed

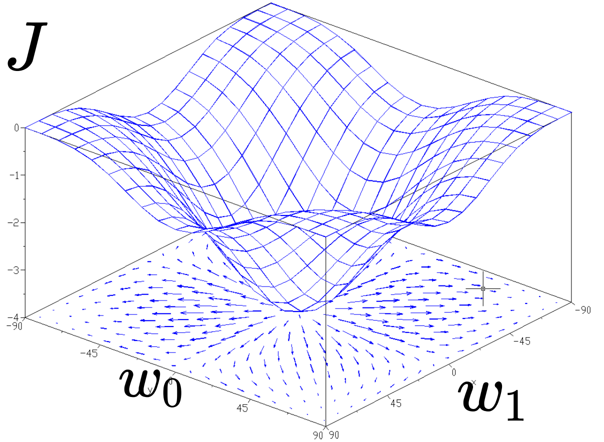
$$\frac{\partial}{\partial w_1} J(w_0, w_1) \triangleq \lim_{\epsilon \to 0} \frac{J(w_0, w_1+\epsilon) - J(w_0, w_1)}{\epsilon}$$

critical point: all partial derivatives are zero

gradient: vector of all partial derivatives

$$\nabla J(w) = [\frac{\partial}{\partial w_1} J(w), \cdots \frac{\partial}{\partial w_D} J(w)]^\top$$

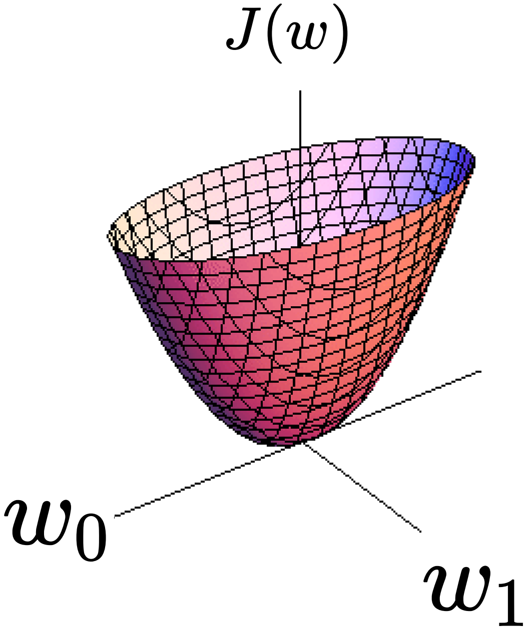




Finding w (any D)

setting $\frac{\partial}{\partial w_i} J(w) = 0$

$$\frac{\partial}{\partial w_i} \sum_n \frac{1}{2}(y^{(n)} - f_w(x^{(n)}))^2 = 0$$



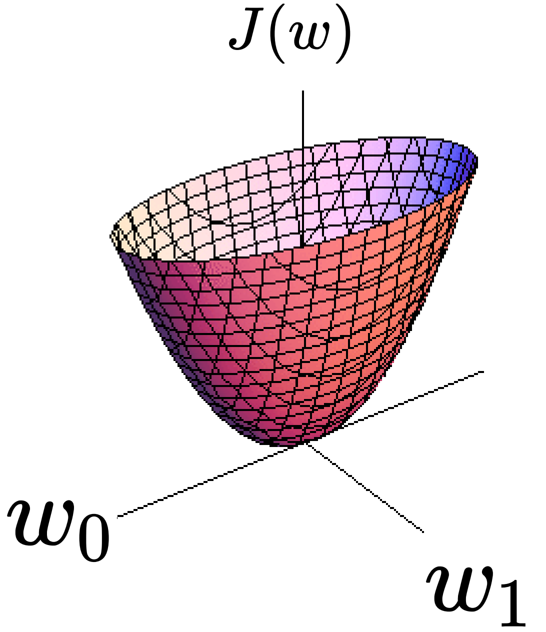
cost is a smooth and convex function of w

Finding w (any D)

setting $\frac{\partial}{\partial w_i} J(w) = 0$

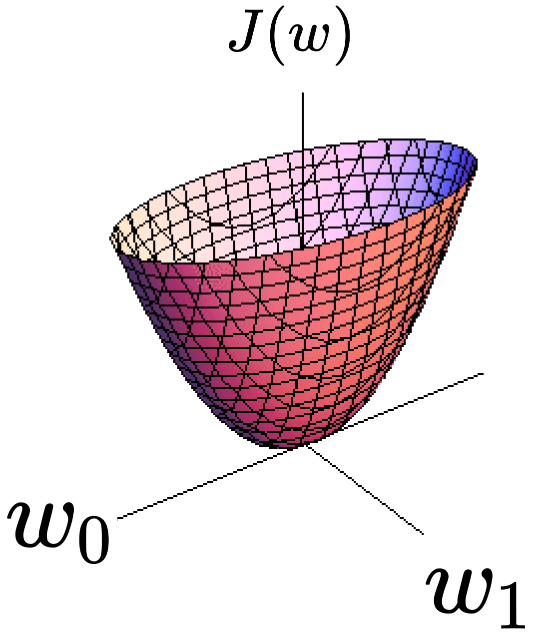
$$\frac{\partial}{\partial w_i} \sum_n \frac{1}{2}(y^{(n)} - f_w(x^{(n)}))^2 = 0$$

using **chain rule**: $\frac{\partial J}{\partial w_i} = \frac{\mathrm{d}J}{\mathrm{d}f_w} \frac{\partial f_w}{\partial w_i}$



cost is a smooth and convex function of w

Finding w (any D)



cost is a smooth and convex function of w

setting $\frac{\partial}{\partial w_i} J(w) = 0$

$$\frac{\partial}{\partial w_i} \sum_n \frac{1}{2}\left(y^{(n)} - f_w(x^{(n)})\right)^2 = 0$$

using **chain rule**: $\frac{\partial J}{\partial w_i} = \frac{\mathrm{d}J}{\mathrm{d}f_w} \frac{\partial f_w}{\partial w_i}$

we get $\sum_n (w^\top x^{(n)} - y^{(n)}) x_d^{(n)} = 0 \quad \forall d \in \{1, \ldots, D\}$

Normal equation

system of D linear equations

$$\sum_n (y^{(n)} - w^\top x^{(n)}) x_d^{(n)} = 0 \quad \forall d$$

Normal equation

system of D linear equations

$$\sum_n (y^{(n)} - w^\top x^{(n)}) x_d^{(n)} = 0 \quad \forall d$$

matrix form (using the design matrix)

each row enforces one of D equations

$$\overset{D \times N}{\overline{X^\top}} \overset{N \times 1}{\overline{(y - Xw)}} = \vec{0}$$

Normal equation

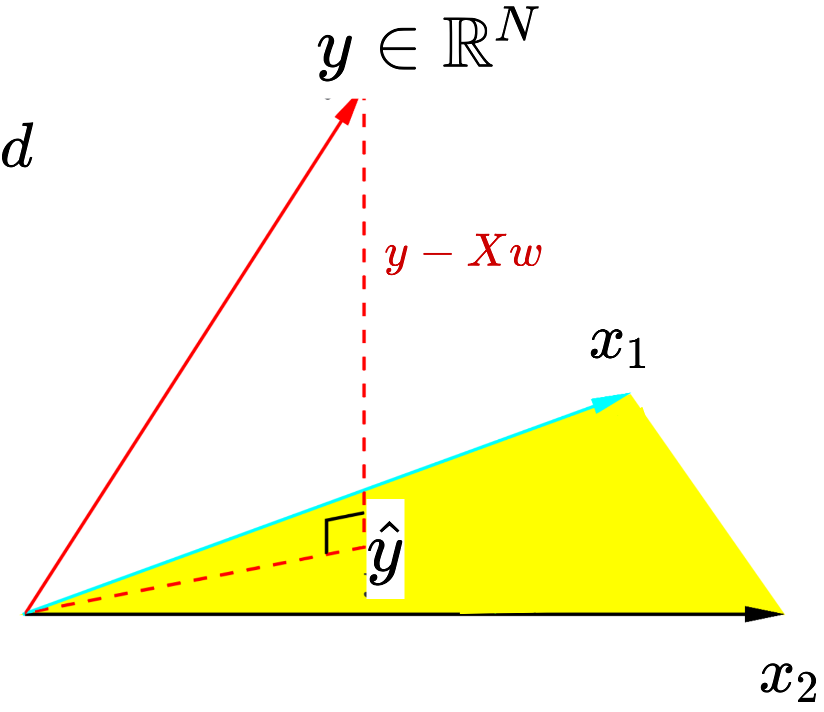
system of D linear equations

$$\sum_n (y^{(n)} - w^\top x^{(n)}) x_d^{(n)} = 0 \quad \forall d$$

matrix form (using the design matrix)

each row enforces one of D equations

$$\overset{D \times N}{X^\top} \overset{N \times 1}{(y - Xw)} = \vec{0}$$



Normal equation

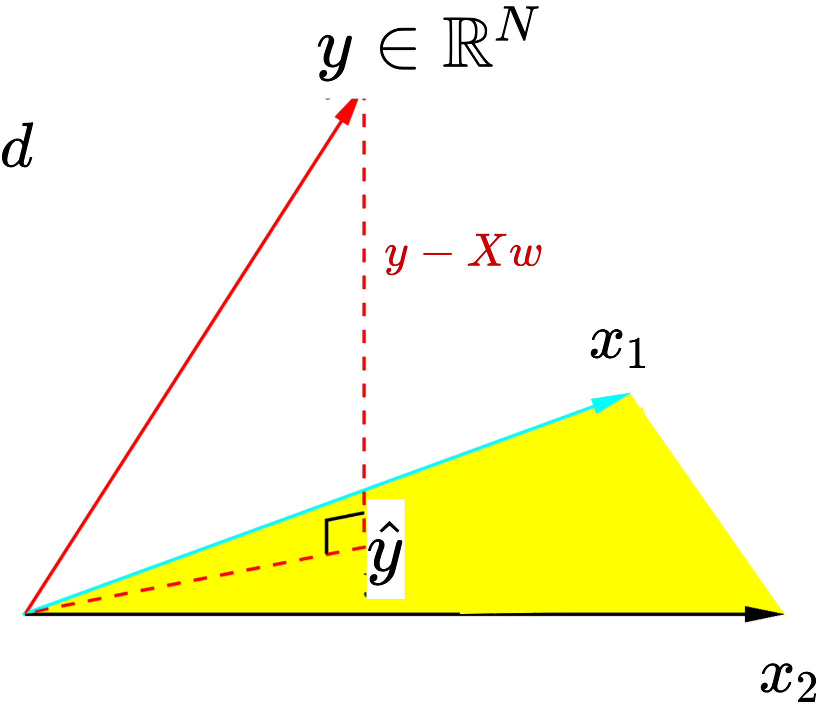
system of D linear equations

$$\sum_n (y^{(n)} - w^\top x^{(n)}) x_d^{(n)} = 0 \quad \forall d$$

matrix form (using the design matrix)

each row enforces one of D equations

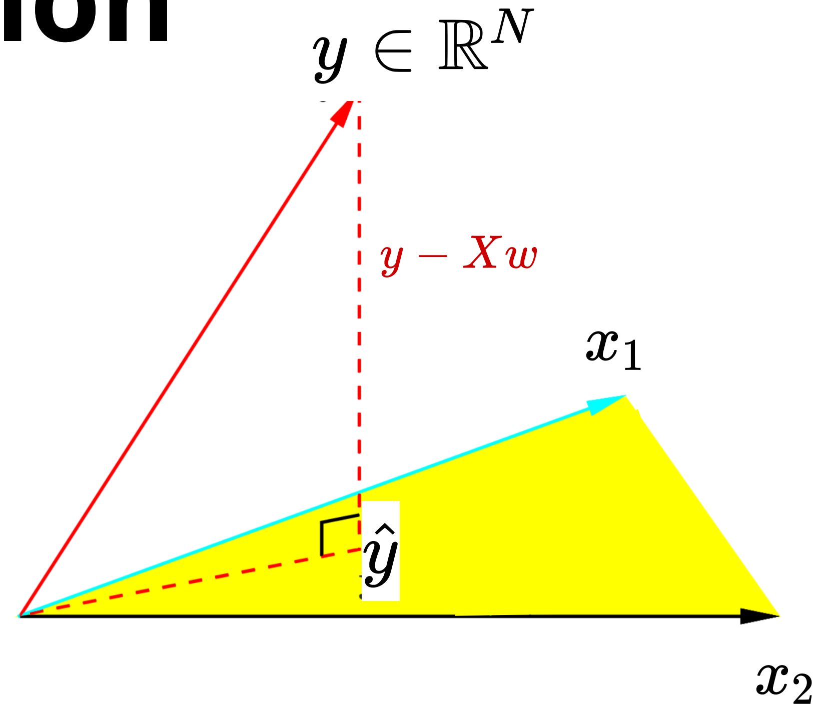
$$\overset{D \times N}{X^\top}\overset{N \times 1}{(y - Xw)} = \vec{0}$$



Normal equation: because for optimal w, the residual vector is normal to column space of the design matrix

Direct solution

we can get a closed form solution!

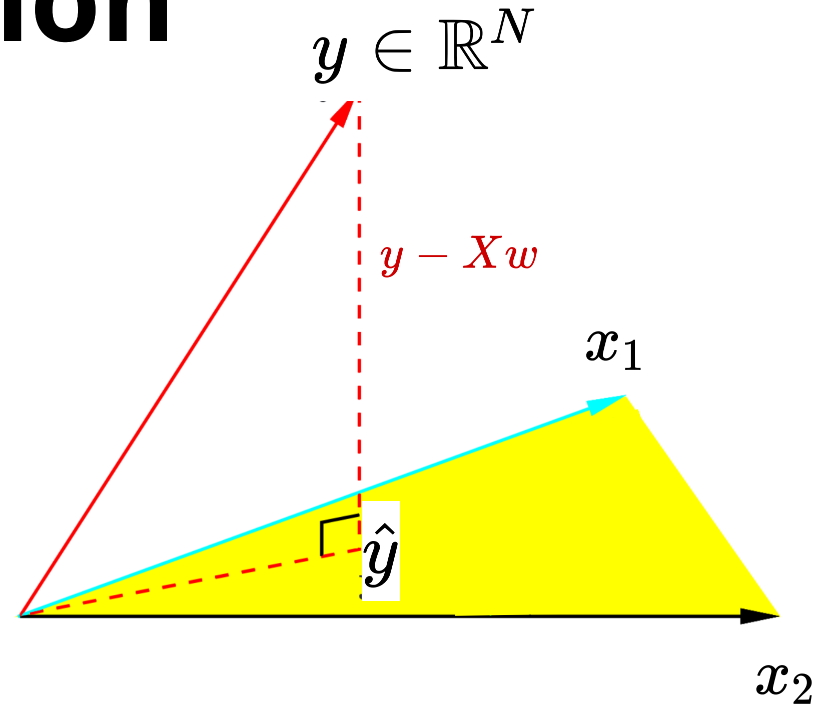


Direct solution

we can get a closed form solution!

$$X^\top(y - Xw) = \vec{0}$$

$$X^\top X w = X^\top y$$



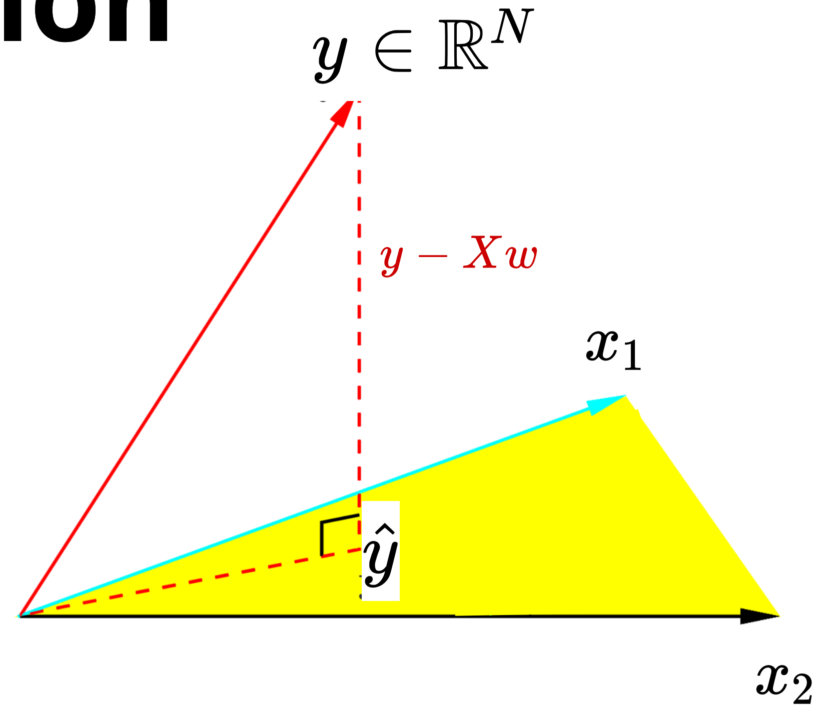
Direct solution

we can get a closed form solution!

$$X^\top(y - Xw) = \vec{0}$$

$$X^\top Xw = X^\top y$$

$$w^* = (X^\top X)^{-1}X^\top y$$



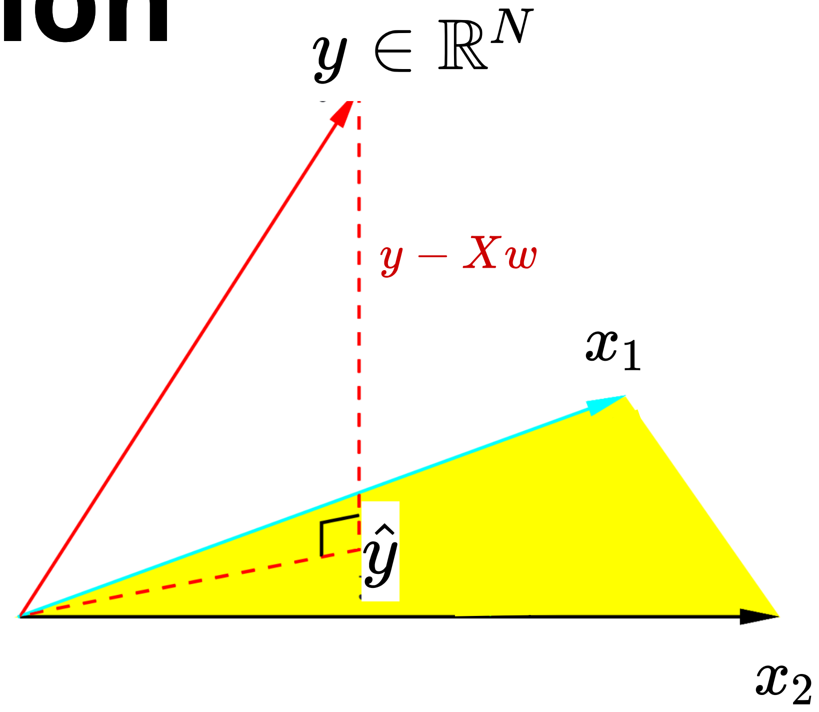
Direct solution

we can get a closed form solution!

$$X^\top (y - Xw) = \vec{0}$$

$$X^\top X w = X^\top y$$

$$w^* = (\underbrace{X^\top X}_{D \times D})^{-1} \underbrace{X^\top}_{D \times N} \underbrace{y}_{N \times 1}$$



Direct solution

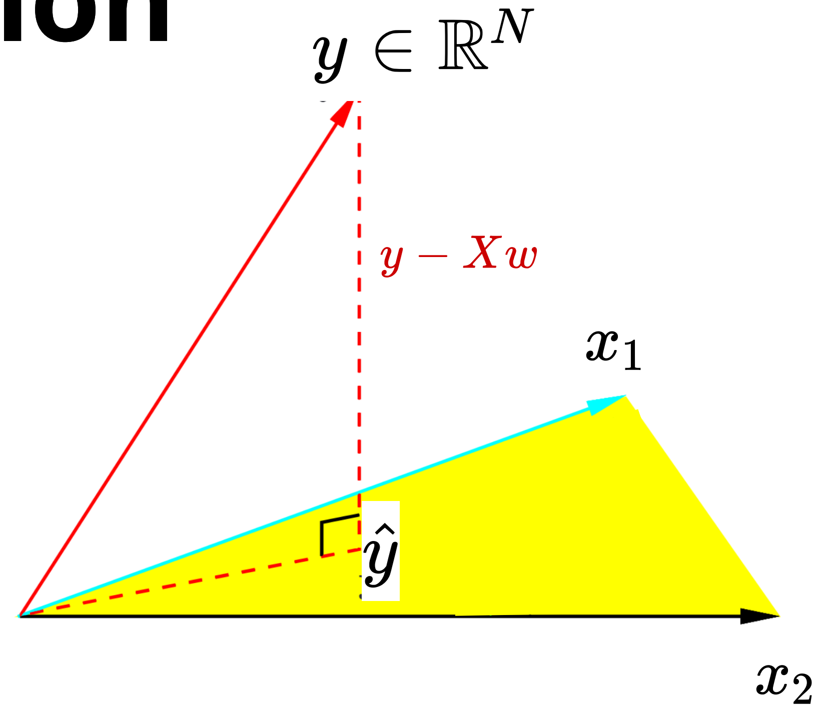
we can get a closed form solution!

$$X^\top (y - Xw) = \vec{0}$$

$$X^\top X w = X^\top y$$

pseudo-inverse of X

$$w^* = \underbrace{(X^\top X)}_{D \times D}{}^{-1} \underbrace{X^\top}_{D \times N} \underbrace{y}_{N \times 1}$$



Direct solution

we can get a closed form solution!

$$X^\top(y - Xw) = \vec{0}$$

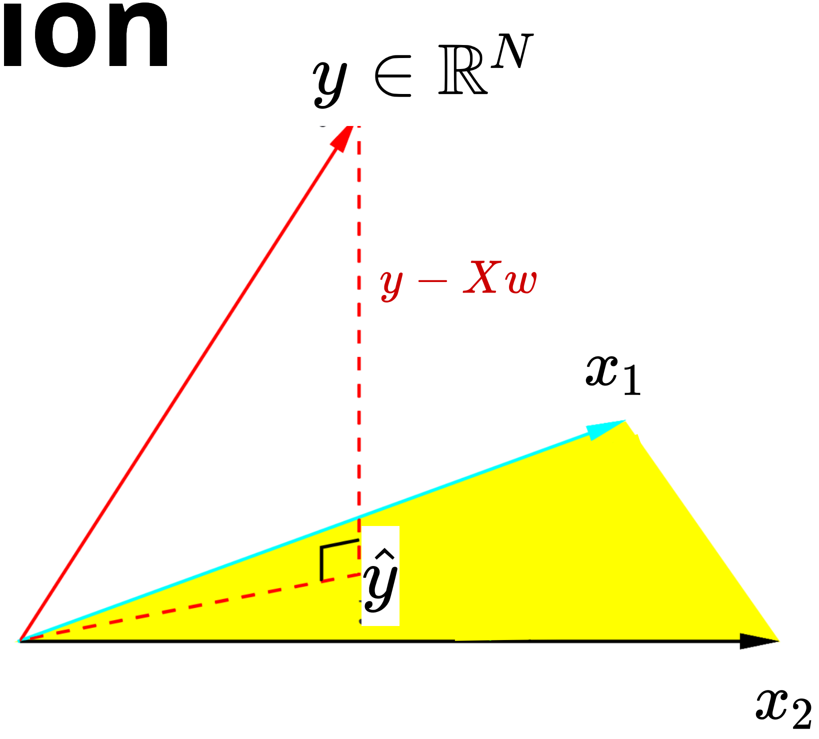
$$X^\top X w = X^\top y$$

pseudo-inverse of X

$$w^* = \underbrace{(X^\top X)}_{D \times D}{}^{-1} \underbrace{X^\top}_{D \times N} \underbrace{y}_{N \times 1}$$

$$\hat{y} = Xw = X(X^\top X)^{-1} X^\top y$$

projection matrix into column space of **X**



Direct solution

we can get a closed form solution!

$$X^\top (y - Xw) = \vec{0}$$

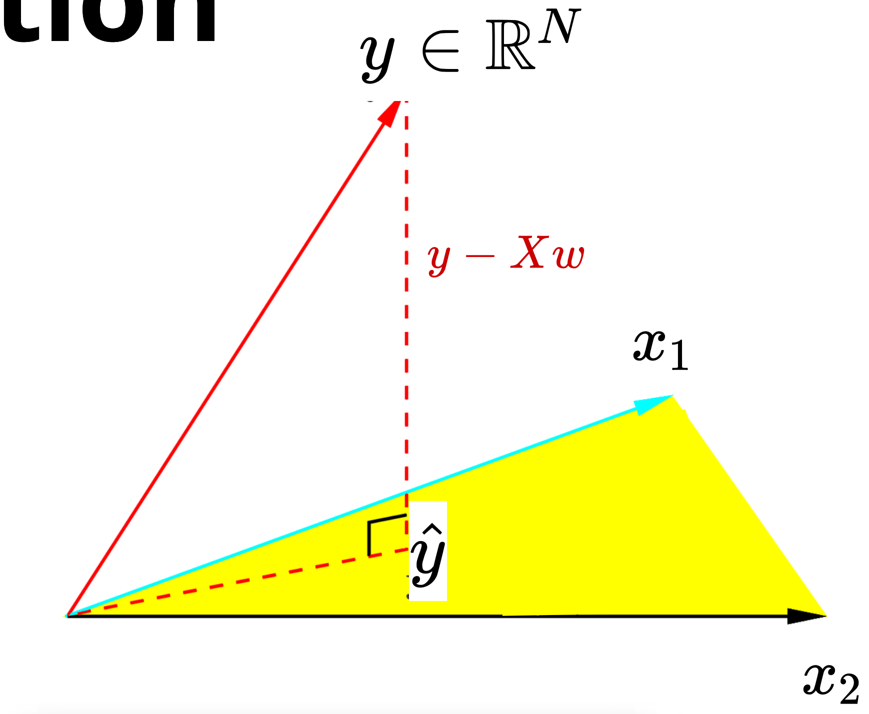
$$X^\top X w = X^\top y$$

pseudo-inverse of X

$$w^* = (\underbrace{X^\top X}_{D \times D})^{-1} \underbrace{X^\top}_{D \times N} \underbrace{y}_{N \times 1}$$

$$\hat{y} = Xw = X(X^\top X)^{-1} X^\top y$$

projection matrix into column space of **X**



```
w = np.linalg.lstsq(X,y)[0]
```

Time complexity

$$w^* = \overset{D \times D}{(X^\top X)}^{-1} \overset{D \times N}{X^\top} \overset{N \times 1}{y}$$

Time complexity

$$w^* = \overset{D \times D}{(X^\top X)}^{-1} \overset{D \times N}{X^\top} \overset{N \times 1}{y}$$

$\mathcal{O}(ND)$ D elements, each using N ops.

Time complexity

$$w^* = (\overset{D \times D}{X^\top X})^{-1} \overset{D \times N}{X^\top} \overset{N \times 1}{y}$$

$\mathcal{O}(ND)$ D elements, each using N ops.

$\mathcal{O}(D^2 N)$ D x D elements, each requiring N multiplications

Time complexity

$$w^* = (\overbrace{X^\top X}^{D \times D})^{-1} \overbrace{X^\top}^{D \times N} \overbrace{y}^{N \times 1}$$

$\mathcal{O}(ND)$ D elements, each using N ops.

$\mathcal{O}(D^3)$ matrix inversion

$\mathcal{O}(D^2 N)$ D x D elements, each requiring N multiplications

Time complexity

$$w^* = (\overbrace{X^\top X}^{D \times D})^{-1} \overbrace{X^\top}^{D \times N} \overbrace{y}^{N \times 1}$$

$\mathcal{O}(ND)$ D elements, each using N ops.

$\mathcal{O}(D^3)$ matrix inversion

$\mathcal{O}(D^2N)$ D x D elements, each requiring N multiplications

total complexity for N > D is $\mathcal{O}(ND^2 + D^3)$

in practice we don't directly use matrix inversion (unstable)

Multiple targets

instead of $y \in \mathbb{R}^N$ we have $Y \in \mathbb{R}^{N \times D'}$

Multiple targets

instead of $y \in \mathbb{R}^N$ we have $Y \in \mathbb{R}^{N \times D'}$

a different weight vectors for each target

each column of Y is associated with a column of W

$$\underset{N \times D'}{\hat{Y}} = \underset{N \times D}{X}\,\underset{D \times D'}{W}$$

Multiple targets

instead of $y \in \mathbb{R}^N$ we have $Y \in \mathbb{R}^{N \times D'}$

a different weight vectors for each target

each column of Y is associated with a column of W

$$\underbrace{\hat{Y}}_{N \times D'} = \underbrace{X}_{N \times D} \underbrace{W}_{D \times D'}$$

$$W^* = (\underbrace{X^\top X}_{D \times D})^{-1} \underbrace{X^\top}_{D \times N} \underbrace{Y}_{N \times D'}$$

Multiple targets

instead of $y \in \mathbb{R}^N$ we have $Y \in \mathbb{R}^{N \times D'}$

a different weight vectors for each target

each column of Y is associated with a column of W

$$\underbrace{\hat{Y}}_{N \times D'} = \underbrace{X}_{N \times D}\underbrace{W}_{D \times D'}$$

$$W^* = (\underbrace{X^\top X}_{D \times D})^{-1}\underbrace{X^\top}_{D \times N}\underbrace{Y}_{N \times D'}$$

```
w = np.linalg.lstsq(X,Y)[0]
```

Nonlinear basis functions

so far we learned a linear function $f_w = \sum_d w_d x_d$

Nonlinear basis functions

so far we learned a linear function $f_w = \sum_d w_d x_d$

nothing changes if we have nonlinear bases $f_w = \sum_d w_d \, \phi_d(x)$

Nonlinear basis functions

so far we learned a linear function $f_w = \sum_d w_d x_d$

nothing changes if we have nonlinear bases $f_w = \sum_d w_d \phi_d(x)$

solution simply becomes $w^* = (\Phi^\top \Phi)^{-1} \Phi^\top y$

replacing X with Φ

$$\Phi = \begin{bmatrix} \phi_1(x^{(1)}), & \phi_2(x^{(1)}), & \cdots, & \phi_D(x^{(1)}) \\ \phi_1(x^{(2)}), & \phi_2(x^{(2)}), & \cdots, & \phi_D(x^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x^{(N)}), & \phi_2(x^{(N)}), & \cdots, & \phi_D(x^{(N)}) \end{bmatrix}$$

Nonlinear basis functions

so far we learned a linear function $f_w = \sum_d w_d x_d$

nothing changes if we have nonlinear bases $f_w = \sum_d w_d \, \phi_d(x)$

solution simply becomes $w^* = (\Phi^\top \Phi)^{-1} \Phi^\top y$

replacing X with Φ

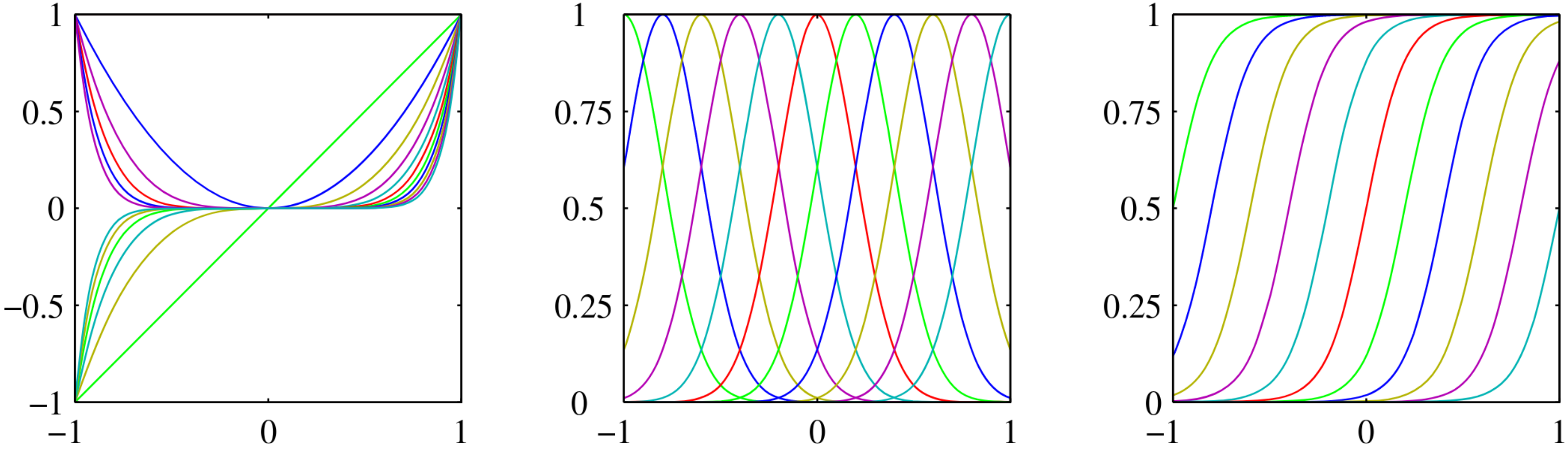
a (nonlinear) feature

$$\Phi = \begin{bmatrix} \phi_1(x^{(1)}), & \phi_2(x^{(1)}), & \cdots, & \phi_D(x^{(1)}) \\ \phi_1(x^{(2)}), & \phi_2(x^{(2)}), & \cdots, & \phi_D(x^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x^{(N)}), & \phi_2(x^{(N)}), & \cdots, & \phi_D(x^{(N)}) \end{bmatrix}$$

one instance

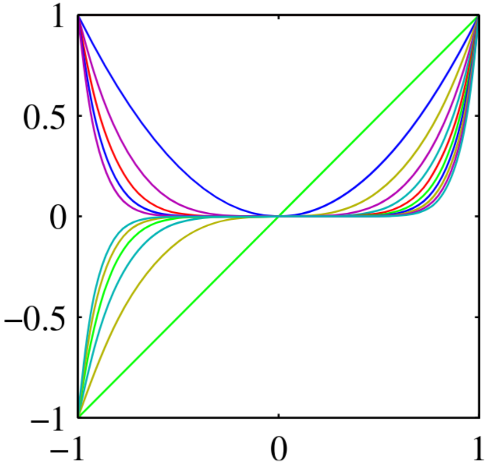
Nonlinear basis functions

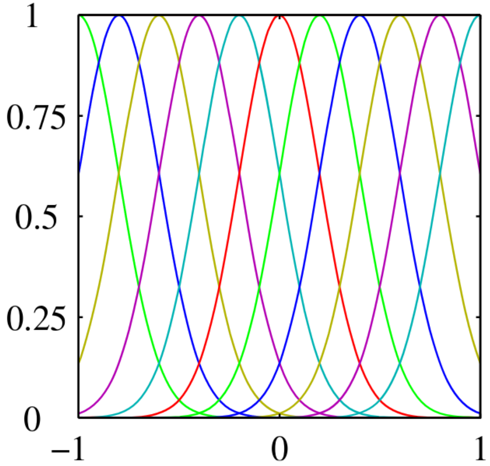
examples original input is scalar $x \in \mathbb{R}$

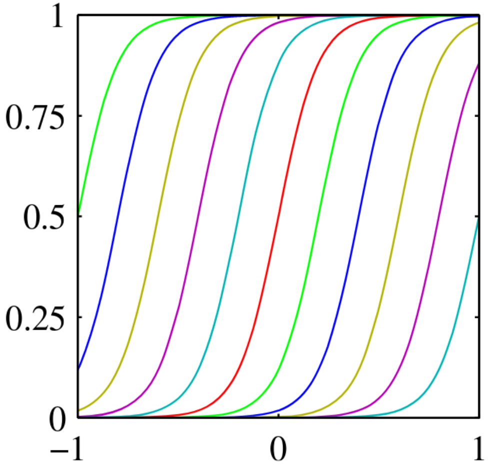


Nonlinear basis functions

examples original input is scalar $x \in \mathbb{R}$





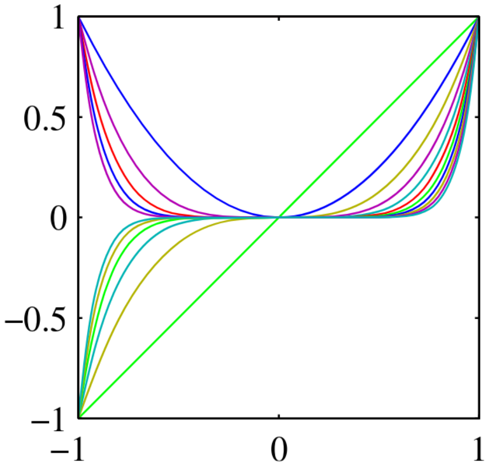


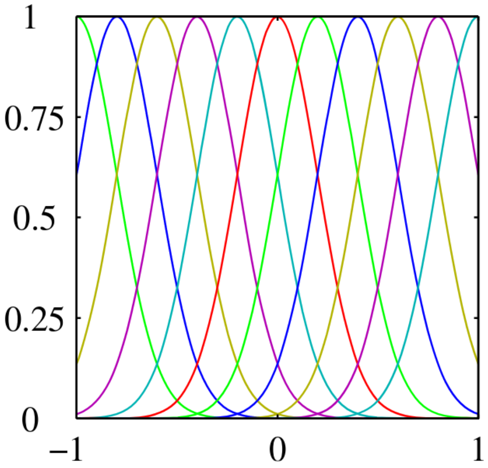
polynomial bases

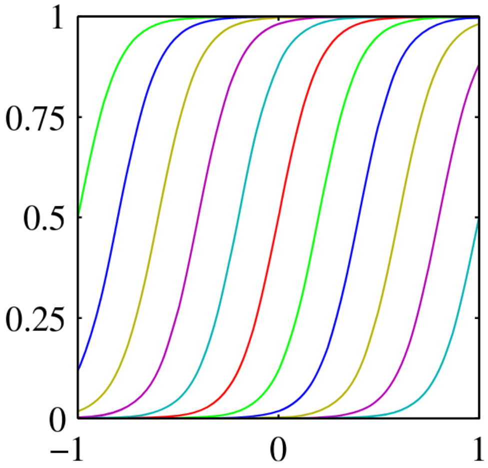
$$\phi_k(x) = x^k$$

Nonlinear basis functions

examples original input is scalar $x \in \mathbb{R}$







polynomial bases

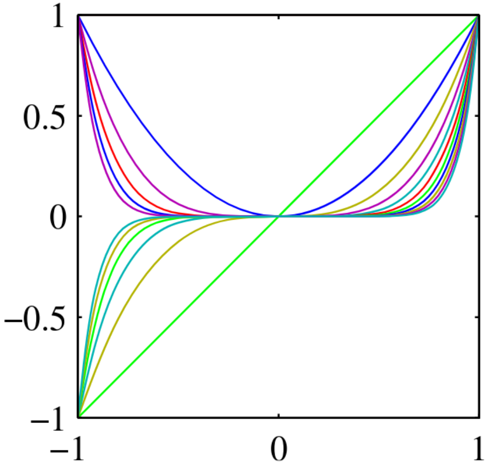
$$\phi_k(x) = x^k$$

Gaussian bases

$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

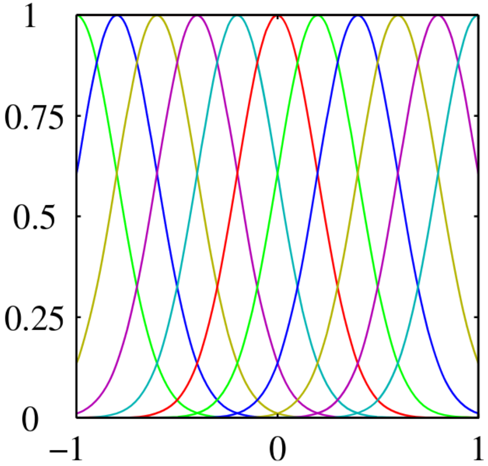
Nonlinear basis functions

examples original input is scalar $x \in \mathbb{R}$



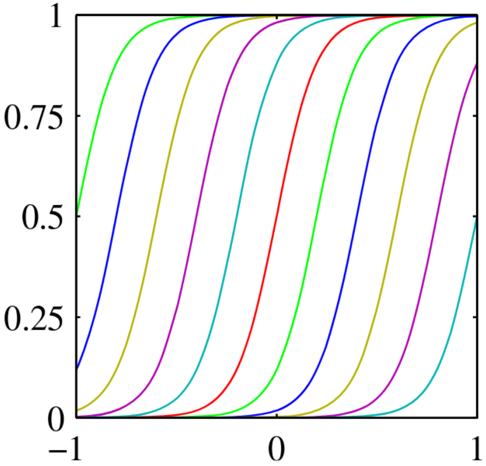
polynomial bases

$$\phi_k(x) = x^k$$



Gaussian bases

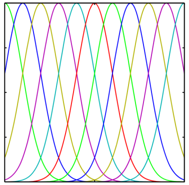
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



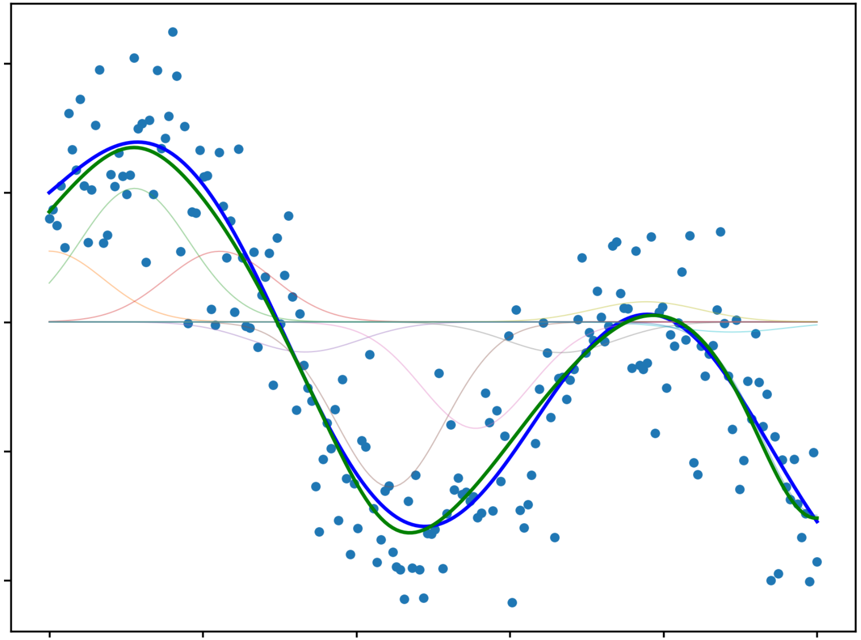
Sigmoid bases

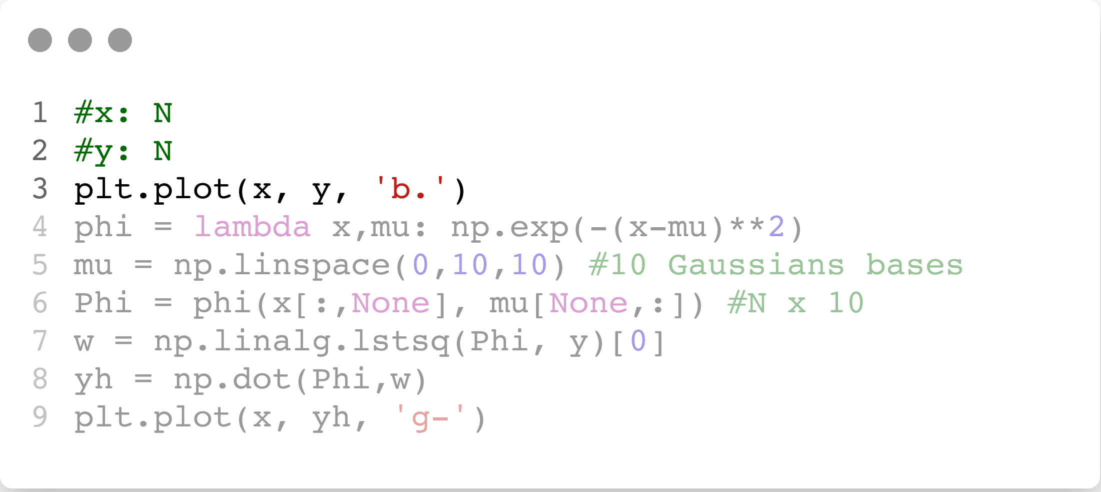
$$\phi_k(x) = \frac{1}{1+e^{-\frac{x-\mu_k}{s}}}$$

Example: Gaussian bases



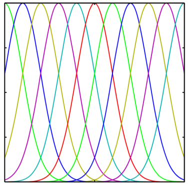
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



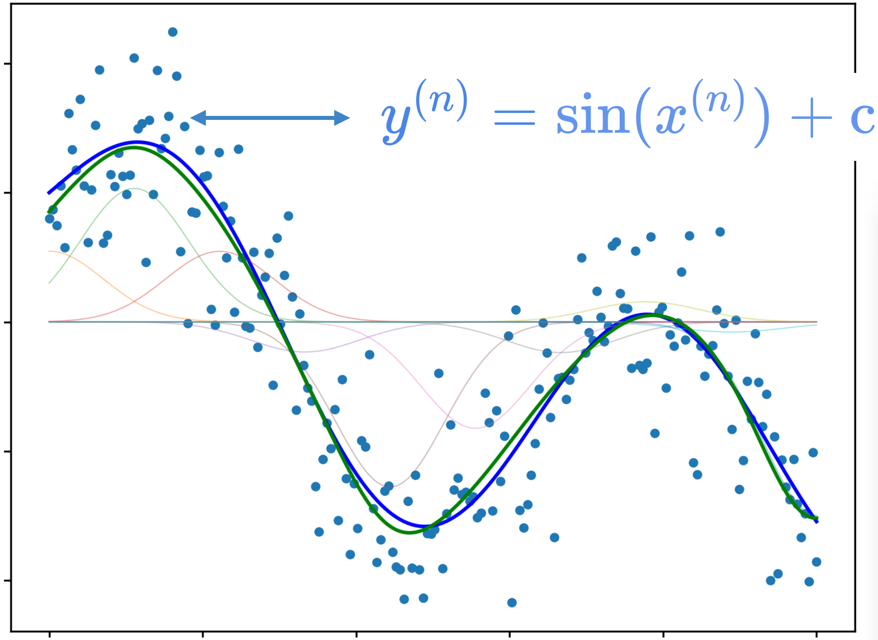


```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

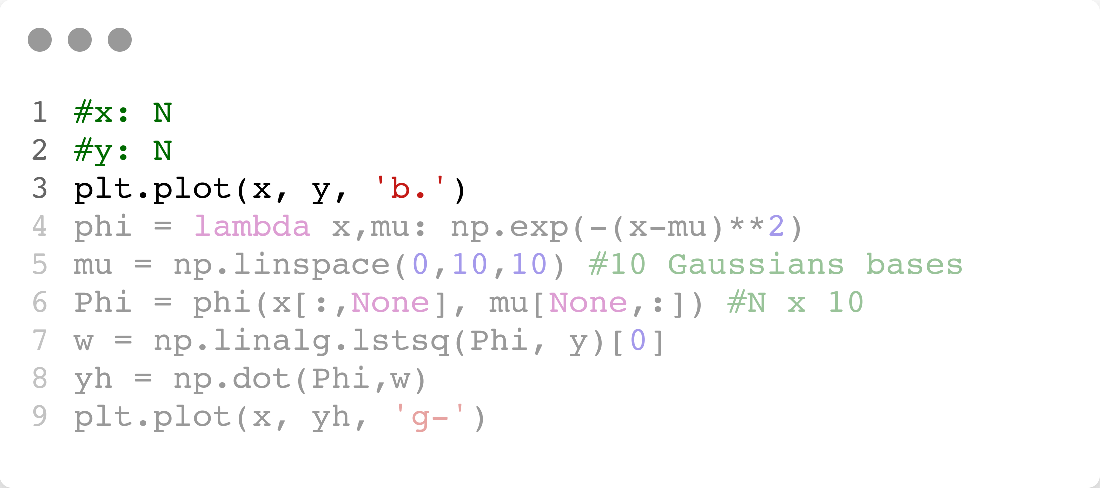
Example: Gaussian bases



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

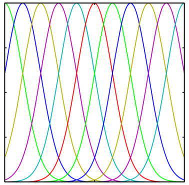


$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

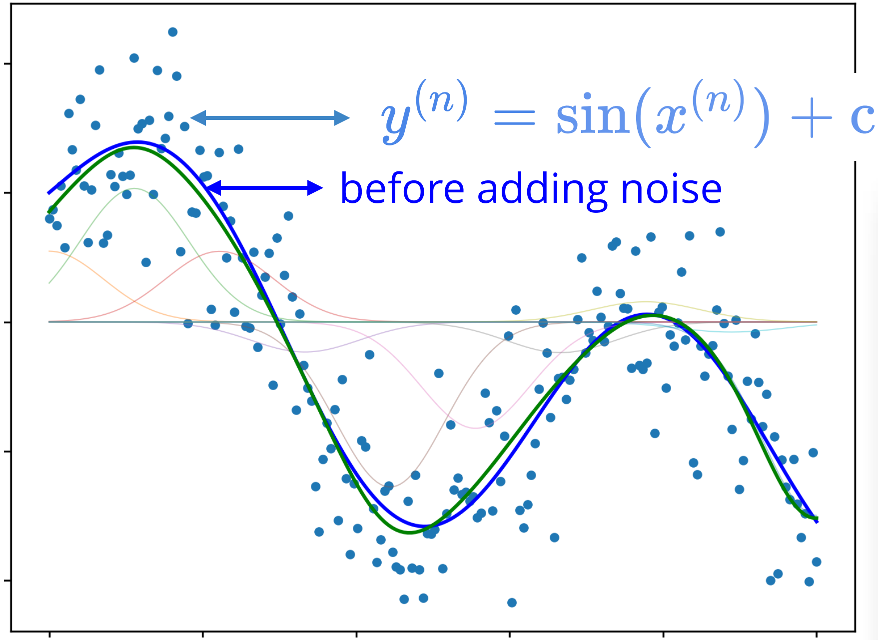


```python
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Example: Gaussian bases



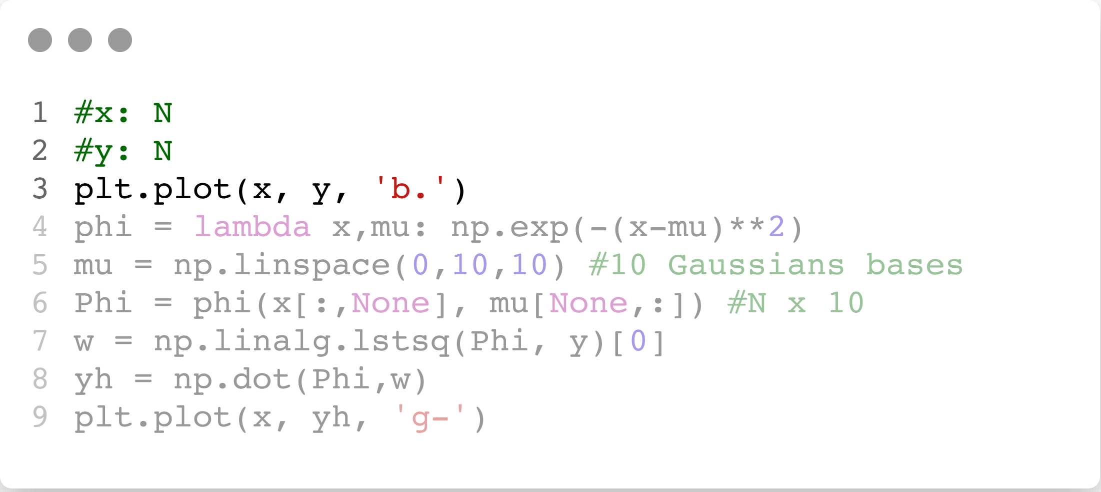
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



noise

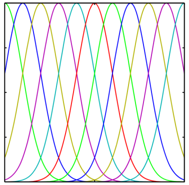
$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \boxed{\epsilon}$$

before adding noise

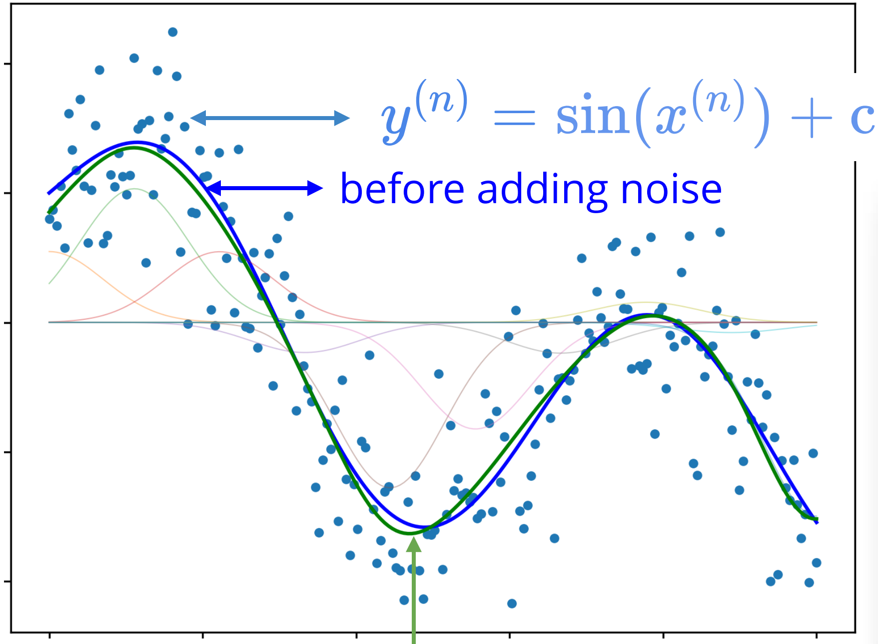


```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Example: Gaussian bases



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

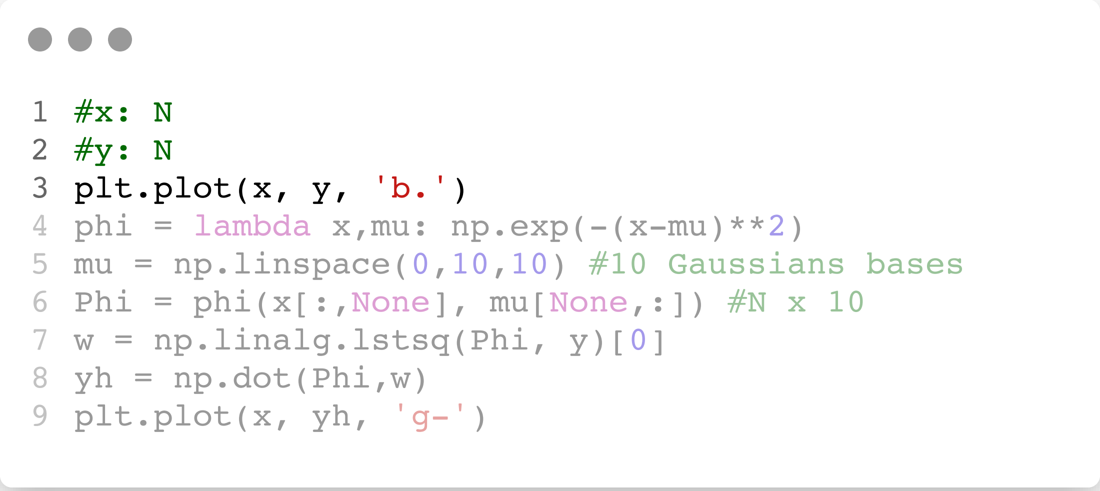


$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

noise

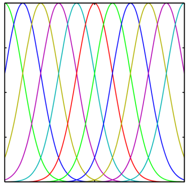
before adding noise

our fit to data using 10 Gaussian bases

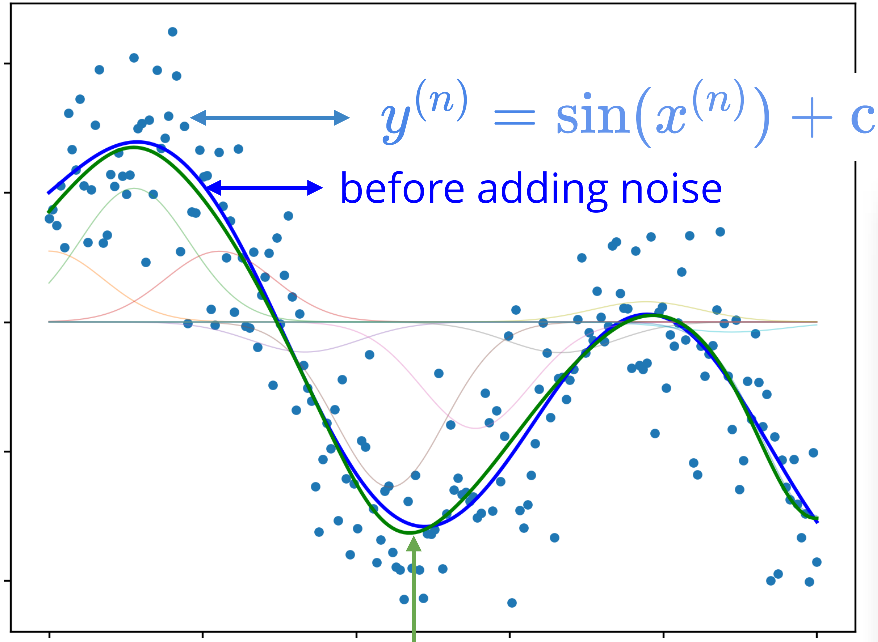


```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Example: Gaussian bases



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

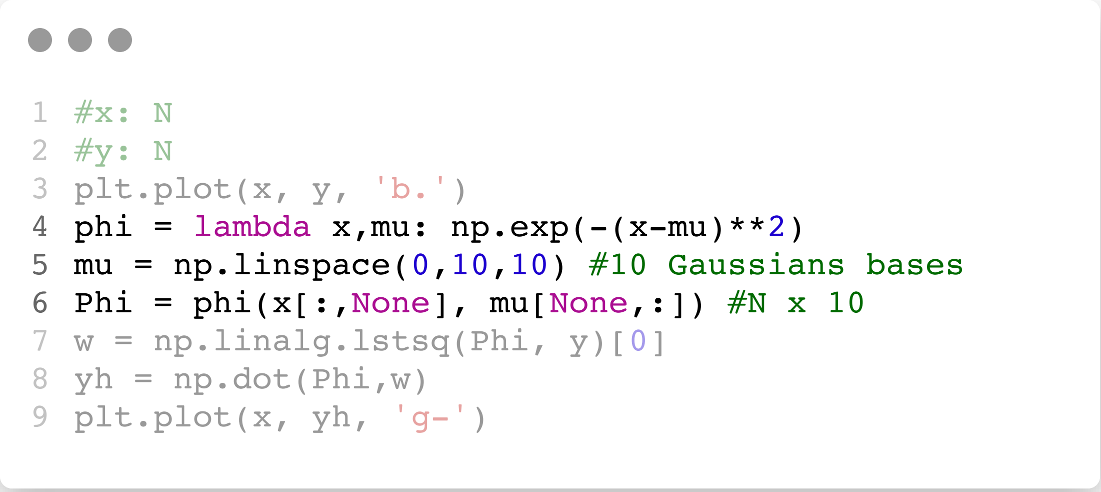


noise

$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

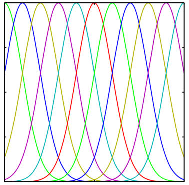
before adding noise

our fit to data using 10 Gaussian bases

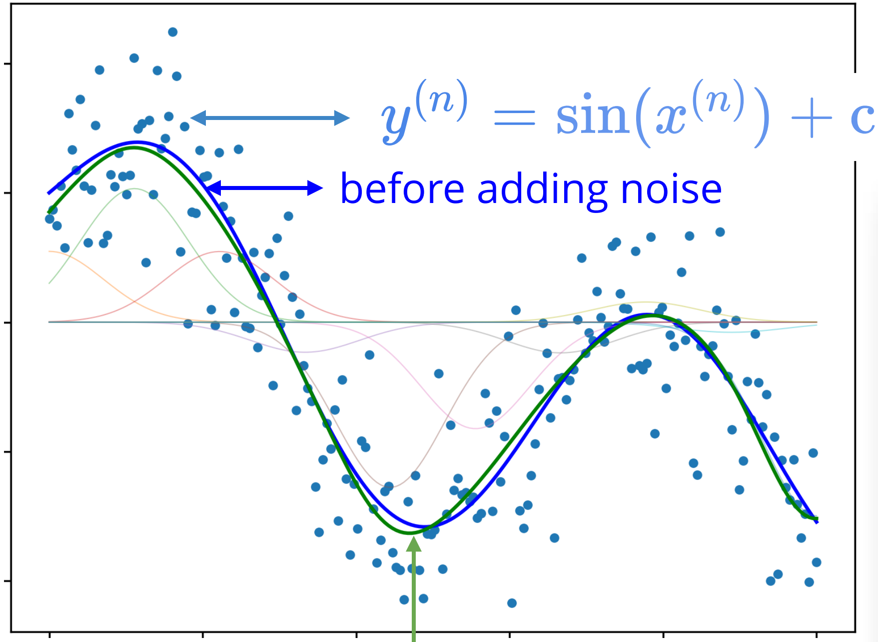


```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Example: Gaussian bases



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

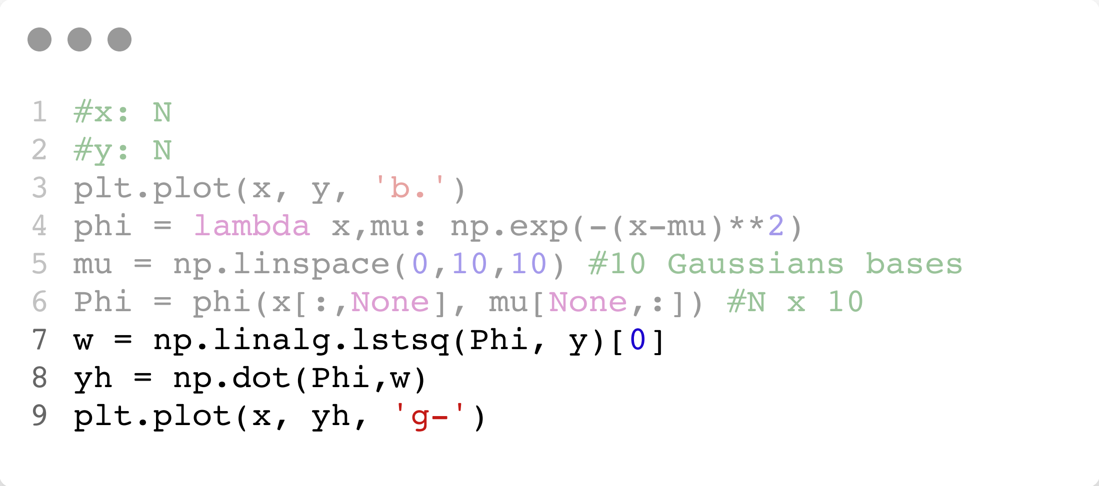


noise

$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

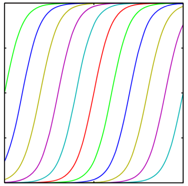
before adding noise

our fit to data using 10 Gaussian bases



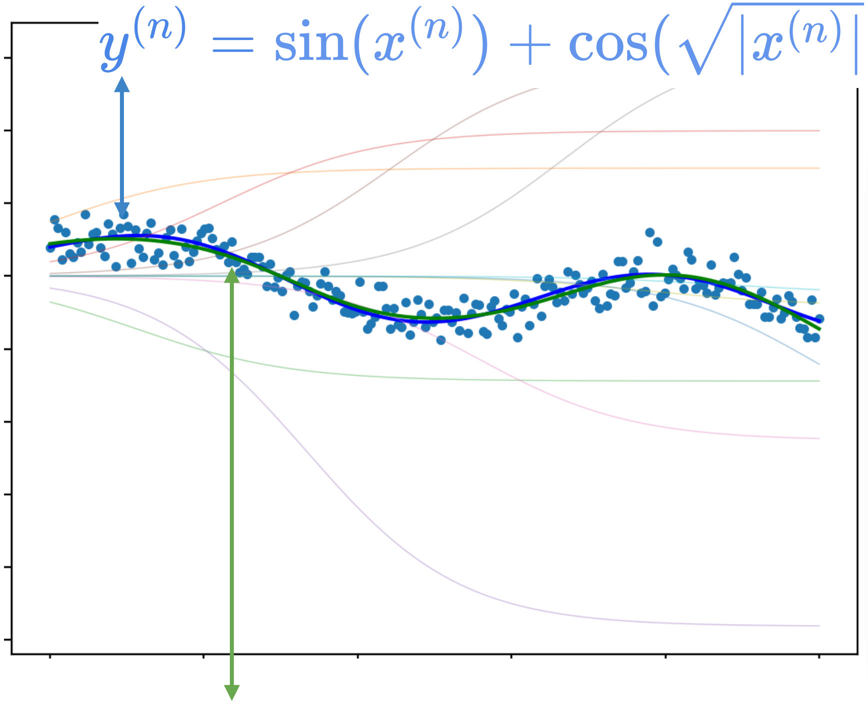
```
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: np.exp(-(x-mu)**2)
mu = np.linspace(0,10,10) #10 Gaussians bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Example: Sigmoid bases



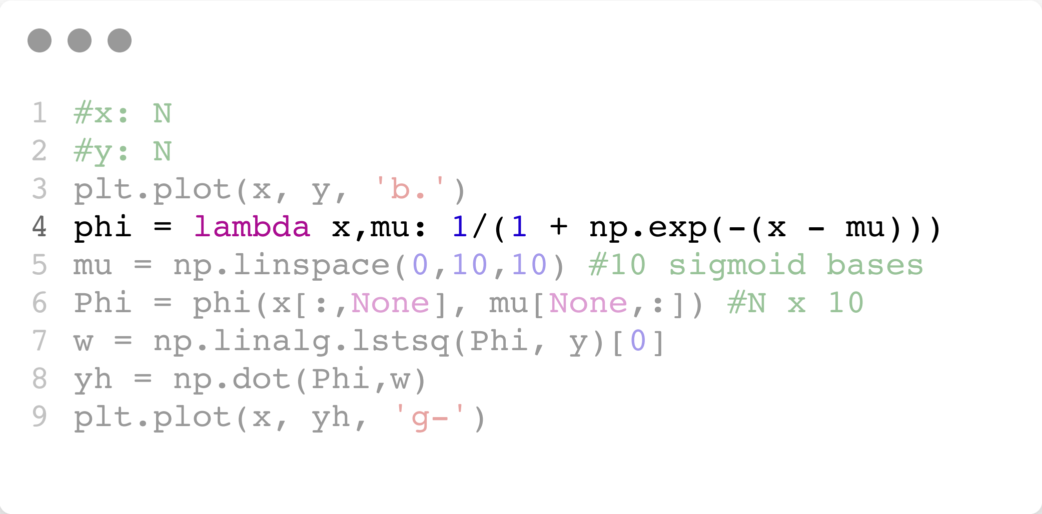
Sigmoid bases

$$\phi_k(x) = \frac{1}{1+e^{-\frac{x-\mu_k}{s}}}$$



$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

our fit to data using 10 sigmoid bases



```python
#x: N
#y: N
plt.plot(x, y, 'b.')
phi = lambda x,mu: 1/(1 + np.exp(-(x - mu)))
mu = np.linspace(0,10,10) #10 sigmoid bases
Phi = phi(x[:,None], mu[None,:]) #N x 10
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```

Problematic settings

In $W^* = (X^\top X)^{-1} X^\top Y$

Problematic settings

In $W^* = (X^\top X)^{-1} X^\top Y$

what if we have a **large dataset**? $N > 100,000,000$

- use stochastic gradient descent

Problematic settings

In $W^* = (X^\top X)^{-1} X^\top Y$

what if we have a **large dataset**? $N > 100,000,000$

- use stochastic gradient descent

what if $X^\top X$ is **not invertible**?

columns of X (features) are not linearly independent

(either redundant features or D > N)

Problematic settings

In $W^* = (X^\top X)^{-1} X^\top Y$

what if we have a **large dataset**? $N > 100,000,000$

- use stochastic gradient descent

what if $X^\top X$ is **not invertible**?

columns of X (features) are not linearly independent
(either redundant features or D > N)

- W* is **not unique, make it unique** by
  - removing redundant features
  - regularization (later!)

Problematic settings

In $W^* = (X^\top X)^{-1} X^\top Y$

what if we have a **large dataset**? $N > 100,000,000$

- use stochastic gradient descent

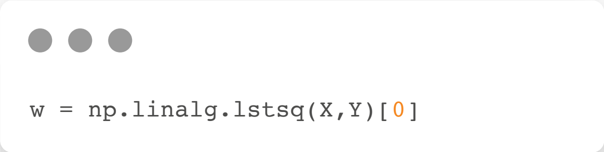
what if $X^\top X$ is **not invertible**?

columns of X (features) are not linearly independent

(either redundant features or D > N)

- W* is **not unique, make it unique** by
  - removing redundant features
  - regularization (later!)

- **or find one** of the solutions
  - decomposition-based (not discussed) methods still work
  - use gradient descent (later!)



```
w = np.linalg.lstsq(X,Y)[0]
```

Summary

linear regression:

- models targets as a linear function of features
- fit the model by minimizing sum of squared errors
- has a direct solution with $\mathcal{O}(ND^2 + D^3)$ complexity
- gradient descent: future

we can build more expressive models:

- using any number of non-linear features
- ensure features are linearly independent