pixel.inria.fr

Least squares for programmers

— with color plates —

Dmitry Sokolov

November 22, 2020

Table of Contents

Coin toss experiment

We conduct n experiments, two events can happen in each one (“success” or “failure”): one happens with probability p, the other one with probability $1-p$.

Coin toss experiment

We conduct n experiments, two events can happen in each one (“success” or “failure”): one happens with probability p, the other one with probability $1 - p$.

The probability of getting exactly k successes in these n experiments

$$P(k; n, p) = C_n^k p^k (1 - p)^{n-k}$$

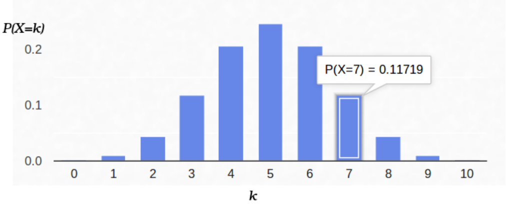
Coin toss experiment

We conduct n experiments, two events can happen in each one ("success" or "failure"): one happens with probability p, the other one with probability $1-p$.

The probability of getting exactly k successes in these n experiments

$$P(k; n, p) = C_n^k p^k (1-p)^{n-k}$$

Toss a coin ten times ($n = 10$), count the number of tails:



an ordinary coin ($p = 1/2$)

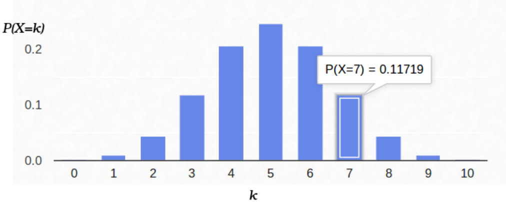
Coin toss experiment

We conduct n experiments, two events can happen in each one ("success" or "failure"): one happens with probability p, the other one with probability $1-p$.

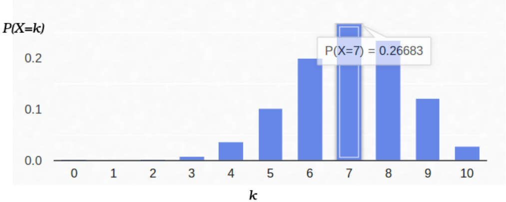
The probability of getting exactly k successes in these n experiments

$$P(k; n, p) = C_n^k p^k (1-p)^{n-k}$$

Toss a coin ten times ($n = 10$), count the number of tails:



an ordinary coin ($p = 1/2$)



a biased coin ($p = 7/10$)

Coin toss: the likelihood function

Suppose we have a real coin, but we do not know p. However, we can toss it ten times. For example, we have counted seven tails. Would it help us to evaluate p?

Coin toss: the likelihood function

Suppose we have a real coin, but we do not know p. However, we can toss it ten times. For example, we have counted seven tails. Would it help us to evaluate p?

Fix $n = 10$ and $k = 7$ in the Bernoulli's formula, leaving p as a free parameter:

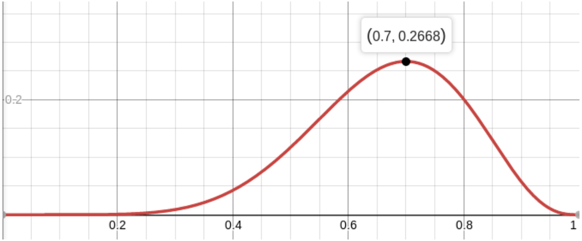
$$\mathcal{L}(p) = C_{10}^{7} p^{7} (1 - p)^{3}$$

Coin toss: the likelihood function

Suppose we have a real coin, but we do not know p. However, we can toss it ten times. For example, we have counted seven tails. Would it help us to evaluate p?

Fix $n = 10$ and $k = 7$ in the Bernoulli's formula, leaving p as a free parameter:

$$\mathcal{L}(p) = C_{10}^{7} p^{7} (1 - p)^{3}$$



N.B. the function is continuous!

Coin toss: the maximum likelihood

Let us solve for $\arg\max_p \mathcal{L}(p) = \arg\max_p \log \mathcal{L}(p)$:

Coin toss: the maximum likelihood

Let us solve for $\arg\max_p \mathcal{L}(p) = \arg\max_p \log \mathcal{L}(p)$:

$$\log \mathcal{L}(p) = \log C_{10}^7 + 7\log p + 3\log(1-p)$$

Coin toss: the maximum likelihood

Let us solve for $\arg\max_p \mathcal{L}(p) = \arg\max_p \log \mathcal{L}(p)$:

$$\log \mathcal{L}(p) = \log C_{10}^{7} + 7\log p + 3\log(1-p)$$

$$\frac{d\log\mathcal{L}}{dp} = \frac{7}{p} - \frac{3}{1-p} = 0$$

Coin toss: the maximum likelihood

Let us solve for $\arg\max_p \mathcal{L}(p) = \arg\max_p \log \mathcal{L}(p)$:

$$\log \mathcal{L}(p) = \log C_{10}^{7} + 7 \log p + 3 \log(1 - p)$$

$$\frac{d \log \mathcal{L}}{dp} = \frac{7}{p} - \frac{3}{1-p} = 0$$

That is, the maximum likelihood (about 27%) is reached at the point $p = 7/10$.

Coin toss: the maximum likelihood

Let us solve for $\arg\max\limits_p \mathcal{L}(p) = \arg\max\limits_p \log \mathcal{L}(p)$:

$$\log \mathcal{L}(p) = \log C_{10}^{7} + 7\log p + 3\log(1-p)$$

$$\frac{d\log\mathcal{L}}{dp} = \frac{7}{p} - \frac{3}{1-p} = 0$$

That is, the maximum likelihood (about 27%) is reached at the point $p = 7/10$.
Just in case, let us check the second derivative:

$$\frac{d^2\log\mathcal{L}}{dp^2} = -\frac{7}{p^2} - \frac{3}{(1-p)^2}$$

At the point $p = 7/10$ it is negative, therefore this point is indeed a maximum of the function $\mathcal{L}$:

$$\frac{d^2\log\mathcal{L}}{dp^2}(0.7) \approx -48 < 0$$

Least squares through maximum likelihood

Let us measure a constant value; all measurements are inherently noisy.

For example, if we measure the battery voltage N times, we get N different measurements:

$$\{U_j\}_{j=1}^{N}$$

Suppose that each measurement U_j is i.i.d. and subject to a Gaussian noise, e.g. it is equal to the real value plus the Gaussian noise. The probability density can be expressed as follows:

$$p(U_j) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(U_j - U)^2}{2\sigma^2}\right),$$

where U is the (unknown) value and σ is the noise amplitude (can be unknown).

Least squares through maximum likelihood

$$\log \mathcal{L}(U, \sigma) = \log \left(\prod_{j=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(U_j - U)^2}{2\sigma^2} \right) \right)$$

Least squares through maximum likelihood

$$\begin{aligned}\log \mathcal{L}(U,\sigma) &= \log\left(\prod_{j=1}^{N} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(U_j-U)^2}{2\sigma^2}\right)\right)\\ &= \sum_{j=1}^{N}\log\left(\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(U_j-U)^2}{2\sigma^2}\right)\right) =\end{aligned}$$

Least squares through maximum likelihood

$$\log \mathcal{L}(U, \sigma) = \log\left(\prod_{j=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(U_j - U)^2}{2\sigma^2}\right)\right)$$

$$= \sum_{j=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(U_j - U)^2}{2\sigma^2}\right)\right) = \sum_{j=1}^{N} \left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{(U_j - U)^2}{2\sigma^2}\right)$$

Least squares through maximum likelihood

$$\begin{aligned}
\log \mathcal{L}(U,\sigma) &= \log\left(\prod_{j=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(U_j-U)^2}{2\sigma^2}\right)\right) \\
&= \sum_{j=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(U_j-U)^2}{2\sigma^2}\right)\right) = \sum_{j=1}^{N}\left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{(U_j-U)^2}{2\sigma^2}\right) \\
&= \underbrace{-N\left(\log\sqrt{2\pi} + \log\sigma\right)}_{\text{does not depend on } \{U_j\}_{j=1}^{N}} - \frac{1}{2\sigma^2}\sum_{j=1}^{N}(U_j-U)^2
\end{aligned}$$

Under Gaussian noise

$$\arg\max_{U} \log \mathcal{L} = \arg\min_{U} \sum_{j=1}^{N}(U_j-U)^2$$

Least squares through maximum likelihood

$$\frac{\partial \log \mathcal{L}}{\partial U} = -\frac{1}{\sigma^2}\sum_{j=1}^{N}(U_j - U) = 0$$

Least squares through maximum likelihood

$$\frac{\partial \log \mathcal{L}}{\partial U} = -\frac{1}{\sigma^2}\sum_{j=1}^{N}(U_j - U) = 0$$

The most plausible estimation of the unknown value U is the simple average of all measurements:

$$U = \frac{\sum_{j=1}^{N} U_j}{N}$$

Least squares through maximum likelihood

$$\frac{\partial \log \mathcal{L}}{\partial U} = -\frac{1}{\sigma^2}\sum_{j=1}^{N}(U_j - U) = 0$$

The most plausible estimation of the unknown value U is the simple average of all measurements:

$$U = \frac{\sum_{j=1}^{N} U_j}{N}$$

And the most plausible estimation of σ turns out to be the standard deviation:

$$\frac{\partial \log \mathcal{L}}{\partial \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{N}(U_j - U)^2 = 0$$

Least squares through maximum likelihood

$$\frac{\partial \log \mathcal{L}}{\partial U} = -\frac{1}{\sigma^2}\sum_{j=1}^{N}(U_j - U) = 0$$

The most plausible estimation of the unknown value U is the simple average of all measurements:

$$U = \frac{\sum_{j=1}^{N} U_j}{N}$$

And the most plausible estimation of σ turns out to be the standard deviation:

$$\frac{\partial \log \mathcal{L}}{\partial \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{N}(U_j - U)^2 = 0 \qquad \sigma = \sqrt{\frac{\sum_{j=1}^{N}(U_j - U)^2}{N}}$$

Such a convoluted way to obtain a simple average of all measurements. . .

Linear regression

It is much harder for less trivial examples. Suppose we have N measurements $\{x_j, y_j\}_{j=1}^N$, and we want to fit a straight line onto it.

Linear regression

It is much harder for less trivial examples. Suppose we have N measurements $\{x_j, y_j\}_{j=1}^N$, and we want to fit a straight line onto it.

$$\log \mathcal{L}(a, b, \sigma) = \log\left(\prod_{j=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_j - ax_j - b)^2}{2\sigma^2}\right)\right) =$$

$$= \underbrace{-N\left(\log\sqrt{2\pi} + \log\sigma\right)}_{\text{does not depend on } a,b} - \frac{1}{2\sigma^2} \underbrace{\sum_{j=1}^N (y_j - ax_j - b)^2}_{:=S(a,b)}$$

Linear regression

It is much harder for less trivial examples. Suppose we have N measurements $\{x_j, y_j\}_{j=1}^N$, and we want to fit a straight line onto it.

$$\log \mathcal{L}(a, b, \sigma) = \log \left(\prod_{j=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_j - ax_j - b)^2}{2\sigma^2} \right) \right) =$$

$$= \underbrace{-N\left(\log\sqrt{2\pi} + \log\sigma\right)}_{\text{does not depend on } a,b} - \frac{1}{2\sigma^2} \underbrace{\sum_{j=1}^{N} (y_j - ax_j - b)^2}_{:=S(a,b)}$$

As before, $\arg\max_{a,b} \log \mathcal{L} = \arg\min_{a,b} S(a, b)$.

Linear regression

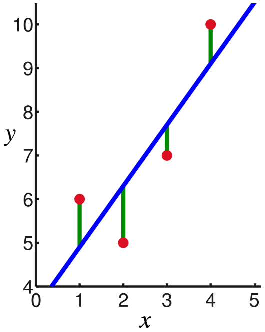
$$S(a, b) := \sum_{j=1}^{N} (y_j - a x_j - b)^2$$

Linear regression

$$S(a, b) := \sum_{j=1}^{N} (y_j - a x_j - b)^2$$

$$\frac{\partial S}{\partial a} = \sum_{j=1}^{N} 2 x_j (a x_j + b - y_j) = 0$$

$$\frac{\partial S}{\partial b} = \sum_{j=1}^{N} 2 (a x_j + b - y_j) = 0$$



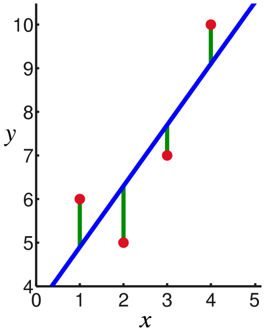
Linear regression

$$S(a, b) := \sum_{j=1}^{N} (y_j - a x_j - b)^2$$

$$\frac{\partial S}{\partial a} = \sum_{j=1}^{N} 2 x_j (a x_j + b - y_j) = 0$$

$$\frac{\partial S}{\partial b} = \sum_{j=1}^{N} 2 (a x_j + b - y_j) = 0$$

$$a = \frac{N \sum_{j=1}^{N} x_j y_j - \sum_{j=1}^{N} x_j \sum_{j=1}^{N} y_j}{N \sum_{j=1}^{N} x_j^2 - \left(\sum_{j=1}^{N} x_j \right)^2}$$



$$b = \frac{1}{N} \left(\sum_{j=1}^{N} y_j - a \sum_{j=1}^{N} x_j \right)$$

The takeaway message

The least squares method is a particular case of maximizing likelihood in cases where the probability density is Gaussian.

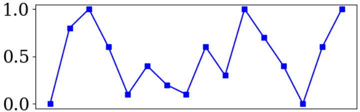
The more we parameters we have, the more cumbersome the analytical solutions are. Fortunately, we are not living in XVIII century anymore, we have computers!

Next we will try to build a geometric intuition on least squares, and see how can least squares problems be efficiently implemented.

Table of Contents

Smooth an array

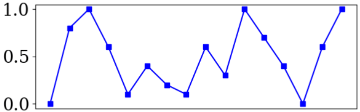
```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```



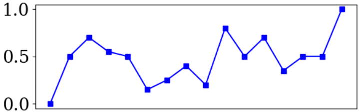
Initialization

Smooth an array

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```



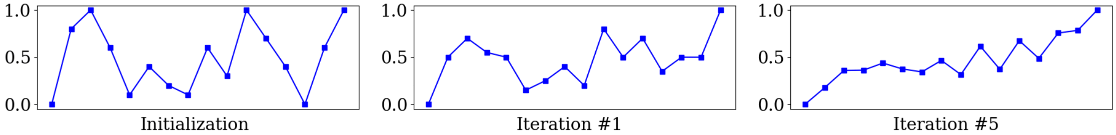
Initialization



Iteration #1

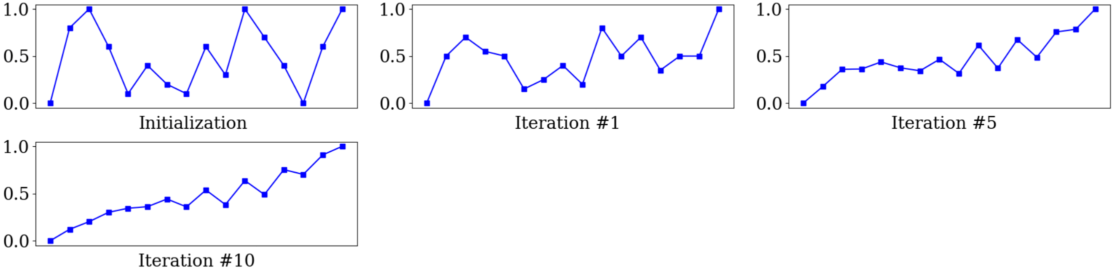
Smooth an array

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```



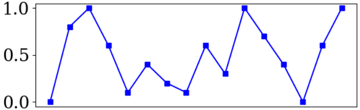
Smooth an array

```python
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```

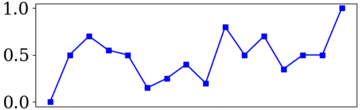


Smooth an array

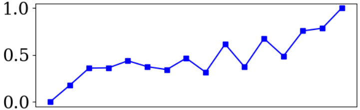
```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```



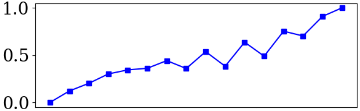
Initialization



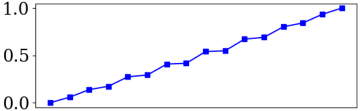
Iteration #1



Iteration #5



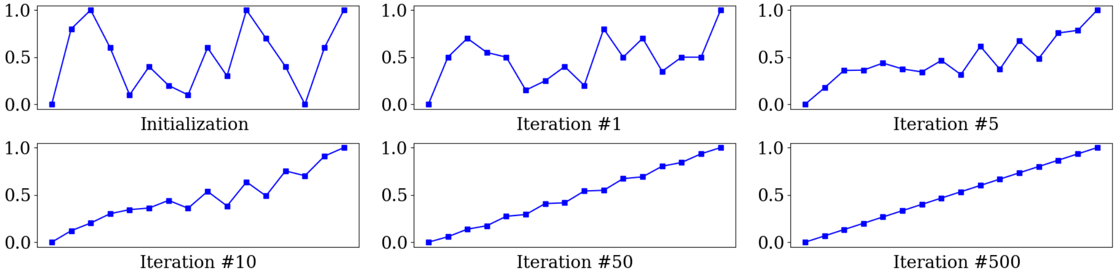
Iteration #10



Iteration #50

Smooth an array

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```



The Jacobi iterative method

Given an ordinary system of linear equations:

$$\begin{cases} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = b_2 \\ & & & \vdots & & & & \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = b_n \end{cases}$$

The Jacobi iterative method

Given an ordinary system of linear equations:

$$\begin{cases} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = b_2 \\ & & & \vdots & & & & \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = b_n \end{cases}$$

Let us rewrite it as follows:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n) \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n) \\ &\vdots \\ x_n &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}) \end{aligned}$$

The Jacobi iterative method

Let us start with an arbitrary vector $\vec{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}\right)$,

The Jacobi iterative method

Let us start with an arbitrary vector $\vec{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$,
we can define $\vec{x}^{(1)}$ as follows:

$$
\begin{aligned}
x_1^{(1)} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(0)} - a_{13}x_3^{(0)} - \cdots - a_{1n}x_n^{(0)}) \\
x_2^{(1)} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(0)} - a_{23}x_3^{(0)} - \cdots - a_{2n}x_n^{(0)}) \\
&\vdots \\
x_n^{(1)} &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(0)} - a_{n2}x_2^{(0)} - \cdots - a_{n,n-1}x_{n-1}^{(0)})
\end{aligned}
$$

The Jacobi iterative method

Let us start with an arbitrary vector $\vec{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$,
we can define $\vec{x}^{(1)}$ as follows:

$$
\begin{aligned}
x_1^{(1)} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(0)} - a_{13}x_3^{(0)} - \cdots - a_{1n}x_n^{(0)}) \\
x_2^{(1)} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(0)} - a_{23}x_3^{(0)} - \cdots - a_{2n}x_n^{(0)}) \\
&\vdots \\
x_n^{(1)} &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(0)} - a_{n2}x_2^{(0)} - \cdots - a_{n,n-1}x_{n-1}^{(0)})
\end{aligned}
$$

Repeating the process k times, the solution can be approximated by the vector $\vec{x}^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right)$.

Back to the array smoothing

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```

Back to the array smoothing

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```

$$x_i = \frac{x_{i-1} + x_{i+1}}{2}$$

Back to the array smoothing

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```

$$x_i = \frac{x_{i-1} + x_{i+1}}{2}$$

$$\Updownarrow$$

$$x_i - x_{i-1} = x_{i+1} - x_i$$

Back to the array smoothing

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```

$$x_i = \frac{x_{i-1} + x_{i+1}}{2}$$

$$\Updownarrow$$

$$x_i - x_{i-1} = x_{i+1} - x_i$$

$$\begin{cases} x_0 &= 0 \\ x_1 - x_0 &= x_2 - x_1 \\ x_2 - x_1 &= x_3 - x_1 \\ &\vdots \\ x_{13} - x_{12} &= x_{14} - x_{13} \\ x_{14} - x_{13} &= x_{15} - x_{14} \\ x_{15} &= 1 \end{cases}$$

The Gauß-Seidel iterative method

Jacobi:

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k-1)} \right), \quad \text{for } i = 1, 2, \dots, n$$

The Gauß-Seidel iterative method

Jacobi:

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k-1)} \right), \quad \text{for } i = 1, 2, \dots, n$$

Gauß-Seidel:

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right), \quad \text{for } i = 1, 2, \dots, n$$

The Gauß-Seidel iterative method

Jacobi:

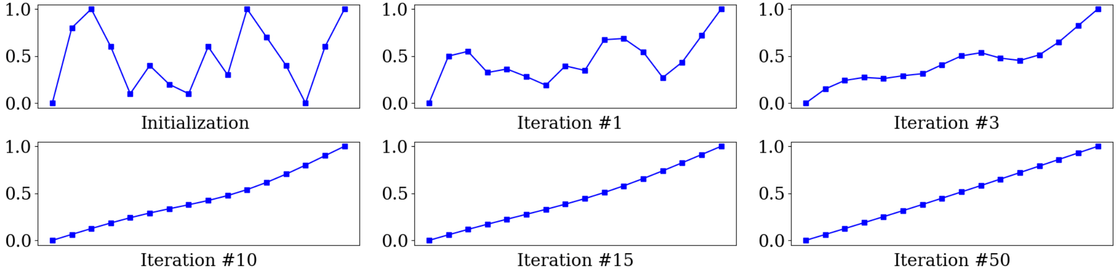
```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    x = [ x[0] ] + \
        [ (x[i-1]+x[i+1])/2. for i in range(1, len(x)-1) ] + \
        [ x[-1] ]
```

Gauß-Seidel:

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```

Smooth an array : Gauß-Seidel

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```



Equality of derivatives vs zero curvature

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```

$$
\begin{cases}
x_0 &= 0 \\
x_1 - x_0 &= x_2 - x_1 \\
x_2 - x_1 &= x_3 - x_1 \\
&\vdots \\
x_{13} - x_{12} &= x_{14} - x_{13} \\
x_{14} - x_{13} &= x_{15} - x_{14} \\
x_{15} &= 1
\end{cases}
$$

Equality of derivatives vs zero curvature

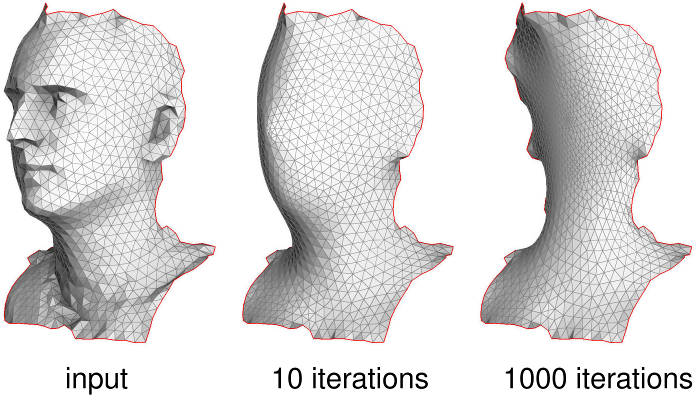
```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```

$$
\left\{
\begin{array}{lllllllll}
x_0 & & & & & & & = 0 \\
-x_0 & +2x_1 & -x_2 & & & & & = 0 \\
 & -x_1 & +2x_2 & -x_3 & & & & = 0 \\
 & & \ddots & \ddots & \ddots & & & \vdots \\
 & & & -x_{12} & +2x_{13} & -x_{14} & & = 0 \\
 & & & & -x_{13} & +2x_{14} & -x_{15} & = 0 \\
 & & & & & & x_{15} & = 1
\end{array}
\right.
$$

It also works for 3d surfaces

```
from mesh import Mesh
import scipy.sparse

m = Mesh("input-face.obj") # load mesh

A = scipy.sparse.lil_matrix((m.nverts, m.nverts))
for v in range(m.nverts): # build a smoothing operator as a sparse matrix
    if m.on_border(v):
        A[v,v] = 1 # fix boundary verts
    else:
        neigh_list = m.neighbors(v)
        for neigh in neigh_list:
            A[v, neigh] = 1/len(neigh_list) # 1-ring barycenter for interior
A = A.tocsr() # sparse row matrix for fast matrix-vector multiplication

for _ in range(8192): # smooth the surface through Gauss-Seidel iterations
    m.V = A.dot(m.V)

print(m) # output smoothed mesh
```

It also works for 3d surfaces

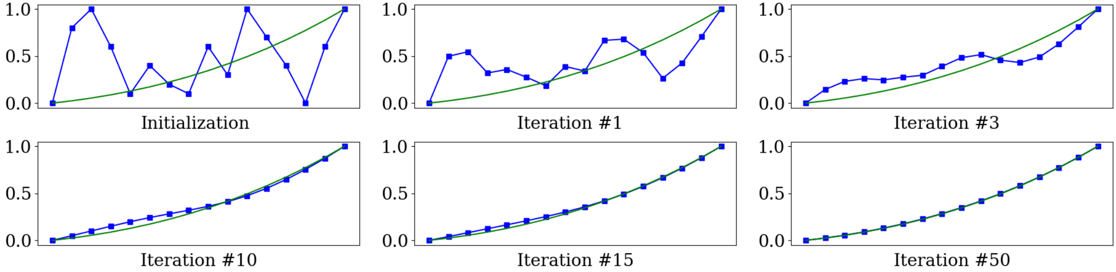


Prescribe the right hand side

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] - (i+15)/15**3 )/2.
```

Prescribe the right hand side

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(512):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] - (i+15)/15**3 )/2.
```



Well duh... Of course it is a cubic polynomial!

The takeaway message

We just saw that mere 3 lines of code can be sufficient to solve a linear system, effectively solving a differential equation.

While it is extremely cool, it raises questions:

- What are the practical consequences for a programmer?
- How do we build these systems?
- Where do we use them?

Table of Contents

Matrices and numbers

What is a number a?

```
float a = 2.7;
```

Matrices and numbers

What is a number *a*?

```
float a = 2.7;
```

Is it a function $y(x) = ax : \mathbb{R} \to \mathbb{R}$?

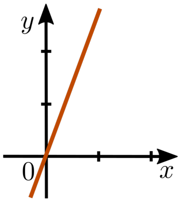
Matrices and numbers

What is a number *a*?

```
float a = 2.7;
```

Is it a function $y(x) = ax : \mathbb{R} \to \mathbb{R}$?

```
float y(float x) {
    return a*x;
}
```



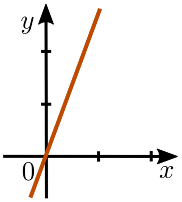
Matrices and numbers

What is a number *a*?

```
float a = 2.7;
```

Is it a function $y(x) = ax : \mathbb{R} \to \mathbb{R}$?

```
float y(float x) {
    return a*x;
}
```



Or is it $y(x) = ax^2 : \mathbb{R} \to \mathbb{R}$?

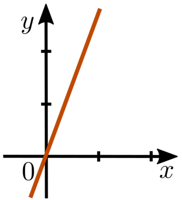
Matrices and numbers

What is a number a?

```
float a = 2.7;
```

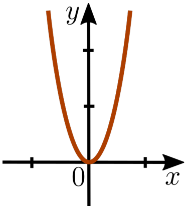
Is it a function $y(x) = ax : \mathbb{R} \to \mathbb{R}$?

```
float y(float x) {
    return a*x;
}
```



Or is it $y(x) = ax^2 : \mathbb{R} \to \mathbb{R}$?

```
float y(float x) {
    return x*a*x;
}
```



Matrices and numbers

The same goes for matrices, what is a matrix $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$?

Matrices and numbers

The same goes for matrices, what is a matrix $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$?

```
float A[2][2] = {{1., .5}, {.5, 1.}};
```

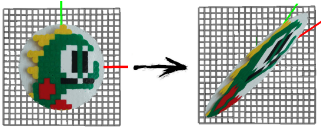
Matrices and numbers

The same goes for matrices, what is a matrix $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$?

```
float A[2][2] = {{1., .5}, {.5, 1.}};
```

Is it $f(x) = Ax : \mathbb{R}^2 \to \mathbb{R}^2$?

```
vector<float> f(vector<float> x) {
  return vector<float>{a11*x[0] + a12*x[1],
                       a21*x[0] + a22*x[1]};
}
```



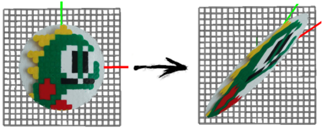
Matrices and numbers

The same goes for matrices, what is a matrix $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$?

```
float A[2][2] = {{1., .5}, {.5, 1.}};
```

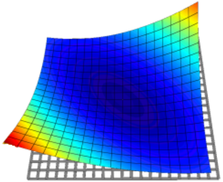
Is it $f(x) = Ax : \mathbb{R}^2 \to \mathbb{R}^2$?

```
vector<float> f(vector<float> x) {
  return vector<float>{a11*x[0] + a12*x[1],
                       a21*x[0] + a22*x[1]};
}
```



Or $f(x) = x^\top Ax = \sum_i \sum_j a_{ij} x_i x_j : \mathbb{R}^2 \to \mathbb{R}$?

```
float f(vector<float> x) {
    return x[0]*a11*x[0] + x[0]*a12*x[1] +
           x[1]*a21*x[0] + x[1]*a22*x[1];
}
```



What is a positive number?

We have a great tool called the predicate “greater than” $>$.

What is a positive number?

We have a great tool called the predicate “greater than” $>$.

Definition

The real number a is positive if and only if for all non-zero real $x \in \mathbb{R},\ x \neq 0$ the condition $ax^2 > 0$ is satisfied.

What is a positive number?

We have a great tool called the predicate “greater than” $>$.

Definition

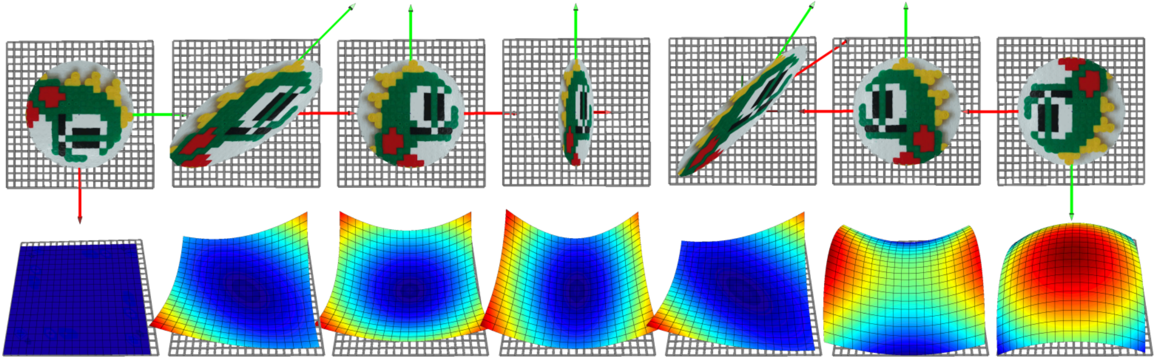
The real number a is positive if and only if for all non-zero real $x \in \mathbb{R},\ x \neq 0$ the condition $ax^2 > 0$ is satisfied.

This definition looks pretty awkward, but it applies perfectly to matrices:

Definition

The square matrix A is called positive definite if for any non-zero x the condition $x^\top Ax > 0$ is met, i.e. the corresponding quadratic form is strictly positive everywhere except at the origin.

What is a positive number?



$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

positive definite

symmetric

Minimizing a 1d quadratic function

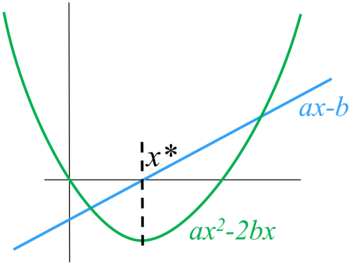
Let us find the minimum of the function $f(x) = ax^2 - 2bx$ (with a positive).

$$\frac{d}{dx} f(x) = 2ax - 2b = 0$$

Minimizing a 1d quadratic function

Let us find the minimum of the function $f(x) = ax^2 - 2bx$ (with a positive).

$$\frac{d}{dx} f(x) = 2ax - 2b = 0$$



In 1d, the solution x^* of the equation $ax - b = 0$ solves the minimization problem $\arg\min_x (ax^2 - 2bx)$ as well.

Differentiating matrix expressions

The first theorem states that 1×1 matrices are invariant w.r.t the transposition:

Theorem

$x \in \mathbb{R} \Rightarrow x^\top = x$

The proof is left as an exercise.

Differentiating matrix expressions

For a 1d function bx we know that $\frac{d}{dx}(bx) = b$, but what happens in the case of a real function of n variables?

Theorem

$\nabla b^\top x = \nabla x^\top b = b$

Differentiating matrix expressions

For a 1d function bx we know that $\frac{d}{dx}(bx) = b$, but what happens in the case of a real function of n variables?

Theorem

$\nabla b^\top x = \nabla x^\top b = b$

$$\nabla(b^\top x) = \begin{bmatrix} \frac{\partial(b^\top x)}{\partial x_1} \\ \vdots \\ \frac{\partial(b^\top x)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial(b_1 x_1 + \dots + b_n x_n)}{\partial x_1} \\ \vdots \\ \frac{\partial(b_1 x_1 + \dots + b_n x_n)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = b$$

Differentiating matrix expressions

For a 1d function ax^2 we know that $\frac{d}{dx}(ax^2) = 2ax$, but what about quadratic forms?

Theorem

$\nabla(x^\top Ax) = (A + A^\top)x$

Note that if A is symmetric, then $\nabla(x^\top Ax) = 2Ax$.

The proof is straightforward, let us express the quadratic form as a double sum:

$$x^\top Ax = \sum_i \sum_j a_{ij} x_i x_j$$

Differentiating matrix expressions

$$\frac{\partial(x^\top Ax)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{k_1}\sum_{k_2} a_{k_1k_2}x_{k_1}x_{k_2}\right) =$$

Differentiating matrix expressions

$$\frac{\partial(x^\top Ax)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{k_1}\sum_{k_2} a_{k_1k_2}x_{k_1}x_{k_2}\right) =$$

$$= \frac{\partial}{\partial x_i}\left(\underbrace{\sum_{k_1\neq i}\sum_{k_2\neq i} a_{k_1k_2}x_{k_1}x_{k_2}}_{k_1\neq i,k_2\neq i} + \underbrace{\sum_{k_2\neq i} a_{ik_2}x_ix_{k_2}}_{k_1=i,k_2\neq i} + \underbrace{\sum_{k_1\neq i} a_{k_1i}x_{k_1}x_i}_{k_1\neq i,k_2=i} + \underbrace{a_{ii}x_i^2}_{k_1=i,k_2=i}\right) =$$

Differentiating matrix expressions

$$\frac{\partial(x^\top Ax)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{k_1}\sum_{k_2} a_{k_1k_2}x_{k_1}x_{k_2}\right) =$$

$$= \frac{\partial}{\partial x_i}\left(\underbrace{\sum_{k_1\neq i}\sum_{k_2\neq i} a_{k_1k_2}x_{k_1}x_{k_2}}_{k_1\neq i,k_2\neq i} + \underbrace{\sum_{k_2\neq i} a_{ik_2}x_ix_{k_2}}_{k_1=i,k_2\neq i} + \underbrace{\sum_{k_1\neq i} a_{k_1i}x_{k_1}x_i}_{k_1\neq i,k_2=i} + \underbrace{a_{ii}x_i^2}_{k_1=i,k_2=i}\right) =$$

$$= \sum_{k_2\neq i} a_{ik_2}x_{k_2} + \sum_{k_1\neq i} a_{k_1i}x_{k_1} + 2a_{ii}x_i =$$

Differentiating matrix expressions

$$\frac{\partial(x^\top Ax)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{k_1}\sum_{k_2} a_{k_1k_2}x_{k_1}x_{k_2}\right) =$$

$$= \frac{\partial}{\partial x_i}\left(\underbrace{\sum_{k_1\neq i}\sum_{k_2\neq i} a_{k_1k_2}x_{k_1}x_{k_2}}_{k_1\neq i,k_2\neq i} + \underbrace{\sum_{k_2\neq i} a_{ik_2}x_ix_{k_2}}_{k_1=i,k_2\neq i} + \underbrace{\sum_{k_1\neq i} a_{k_1i}x_{k_1}x_i}_{k_1\neq i,k_2=i} + \underbrace{a_{ii}x_i^2}_{k_1=i,k_2=i}\right) =$$

$$= \sum_{k_2\neq i} a_{ik_2}x_{k_2} + \sum_{k_1\neq i} a_{k_1i}x_{k_1} + 2a_{ii}x_i =$$

$$= \sum_{k_2} a_{ik_2}x_{k_2} + \sum_{k_1} a_{k_1i}x_{k_1} =$$

Differentiating matrix expressions

$$\frac{\partial(x^\top Ax)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{k_1}\sum_{k_2} a_{k_1k_2}x_{k_1}x_{k_2}\right) =$$

$$= \frac{\partial}{\partial x_i}\left(\underbrace{\sum_{k_1\neq i}\sum_{k_2\neq i} a_{k_1k_2}x_{k_1}x_{k_2}}_{k_1\neq i, k_2\neq i} + \underbrace{\sum_{k_2\neq i} a_{ik_2}x_ix_{k_2}}_{k_1=i,k_2\neq i} + \underbrace{\sum_{k_1\neq i} a_{k_1i}x_{k_1}x_i}_{k_1\neq i,k_2=i} + \underbrace{a_{ii}x_i^2}_{k_1=i,k_2=i}\right) =$$

$$= \sum_{k_2\neq i} a_{ik_2}x_{k_2} + \sum_{k_1\neq i} a_{k_1i}x_{k_1} + 2a_{ii}x_i =$$

$$= \sum_{k_2} a_{ik_2}x_{k_2} + \sum_{k_1} a_{k_1i}x_{k_1} =$$

$$= \sum_j (a_{ij} + a_{ji})x_j \qquad \Rightarrow \nabla(x^\top Ax) = (A + A^\top)x$$

Minimum of a quadratic form and the linear system

Recall that for $a > 0$ solving the equation $ax = b$ is equivalent to the quadratic function $\arg\min\limits_x (ax^2 - 2bx)$ minimization.

Minimum of a quadratic form and the linear system

Recall that for $a > 0$ solving the equation $ax = b$ is equivalent to the quadratic function $\arg\min_x (ax^2 - 2bx)$ minimization.

To minimize a quadratic form $\arg\min_{x \in \mathbb{R}^n} (x^\top A x - 2b^\top x)$ with a symmetric positive definite matrix A, equate the derivative to zero:

$$\nabla(x^\top A x - 2b^\top x) = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^\top.$$

Minimum of a quadratic form and the linear system

Recall that for $a > 0$ solving the equation $ax = b$ is equivalent to the quadratic function $\arg\min_x(ax^2 - 2bx)$ minimization.

To minimize a quadratic form $\arg\min_{x\in\mathbb{R}^n}(x^\top Ax - 2b^\top x)$ with a symmetric positive definite matrix A, equate the derivative to zero:

$$\nabla(x^\top Ax - 2b^\top x) = \begin{bmatrix}0 & \dots & 0\end{bmatrix}^\top.$$

The Hamilton operator is linear: $\nabla(x^\top Ax) - 2\nabla(b^\top x) = \begin{bmatrix}0 & \dots & 0\end{bmatrix}^\top$.

Minimum of a quadratic form and the linear system

Recall that for $a > 0$ solving the equation $ax = b$ is equivalent to the quadratic function $\arg\min_x (ax^2 - 2bx)$ minimization.

To minimize a quadratic form $\arg\min_{x \in \mathbb{R}^n} (x^\top Ax - 2b^\top x)$ with a symmetric positive definite matrix A, equate the derivative to zero:

$$\nabla(x^\top Ax - 2b^\top x) = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^\top.$$

The Hamilton operator is linear: $\nabla(x^\top Ax) - 2\nabla(b^\top x) = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^\top$.
Apply the differentiation theorems:

$$(A + A^\top)x - 2b = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^\top.$$

Minimum of a quadratic form and the linear system

Recall that for $a > 0$ solving the equation $ax = b$ is equivalent to the quadratic function $\arg\min_x (ax^2 - 2bx)$ minimization.

To minimize a quadratic form $\arg\min_{x\in\mathbb{R}^n} (x^\top Ax - 2b^\top x)$ with a symmetric positive definite matrix A, equate the derivative to zero:

$$\nabla(x^\top Ax - 2b^\top x) = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^\top.$$

The Hamilton operator is linear: $\nabla(x^\top Ax) - 2\nabla(b^\top x) = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^\top$.
Apply the differentiation theorems:

$$(A + A^\top)x - 2b = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^\top.$$

Recall that A is symmetric: $Ax = b$.

Back to the linear regression

Given two points (x_1, y_1) and (x_2, y_2), find the line that passes through: $y = \alpha x + \beta$.

Back to the linear regression

Given two points (x_1, y_1) and (x_2, y_2), find the line that passes through: $y = \alpha x + \beta$.

$$\begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \end{cases}$$

Back to the linear regression

Given two points (x_1, y_1) and (x_2, y_2), find the line that passes through: $y = \alpha x + \beta$.

$$\begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \end{cases} \qquad \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}}_{:=A} \underbrace{\begin{bmatrix} \alpha \\ \beta \end{bmatrix}}_{:=x} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{:=b} \qquad \Rightarrow x^* = A^{-1} b$$

Back to the linear regression

Given two points (x_1, y_1) and (x_2, y_2), find the line that passes through: $y = \alpha x + \beta$.

$$\begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \end{cases} \qquad \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}}_{:=A} \underbrace{\begin{bmatrix} \alpha \\ \beta \end{bmatrix}}_{:=x} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{:=b} \qquad \Rightarrow x^* = A^{-1} b$$

Now add a **third** point (x_3, y_3):

$$\begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \\ \alpha x_3 + \beta = y_3 \end{cases}$$

Back to the linear regression

Given two points (x_1, y_1) and (x_2, y_2), find the line that passes through: $y = \alpha x + \beta$.

$$\begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \end{cases} \qquad \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}}_{:=A} \underbrace{\begin{bmatrix} \alpha \\ \beta \end{bmatrix}}_{:=x} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{:=b} \qquad \Rightarrow x^* = A^{-1} b$$

Now add a **third** point (x_3, y_3):

$$\begin{cases} \alpha x_1 + \beta = y_1 \\ \alpha x_2 + \beta = y_2 \\ \alpha x_3 + \beta = y_3 \end{cases} \qquad \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix}}_{:=A\,(3\times 2)} \underbrace{\begin{bmatrix} \alpha \\ \beta \end{bmatrix}}_{:=x\,(2\times 1)} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}}_{:=b\,(3\times 1)}$$

A is rectangular, and thus it is not invertible. Oops!

Back to the linear regression

No biggie, let us rewrite the system:

$$\alpha \underbrace{\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top}_{:=\vec{i}} + \beta \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^\top}_{:=\vec{j}} = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix}^\top$$

Back to the linear regression

No biggie, let us rewrite the system:

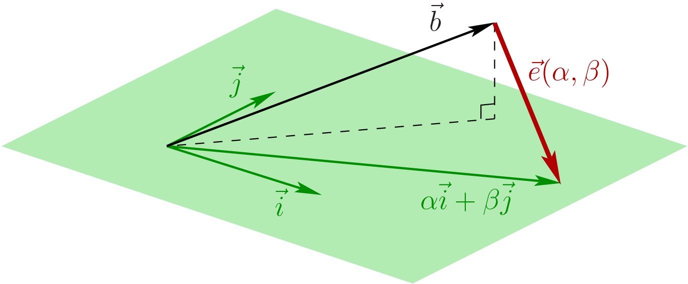
$$\alpha \underbrace{\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top}_{:=\vec{i}} + \beta \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^\top}_{:=\vec{j}} = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix}^\top \qquad \alpha\vec{i} + \beta\vec{j} = \vec{b}$$

Back to the linear regression

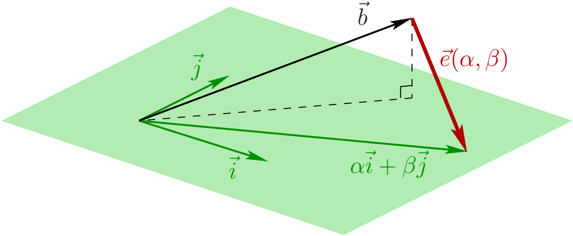
No biggie, let us rewrite the system:

$$\alpha \underbrace{\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^\top}_{:=\vec{i}} + \beta \underbrace{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^\top}_{:=\vec{j}} = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix}^\top \qquad \alpha\vec{i} + \beta\vec{j} = \vec{b}$$

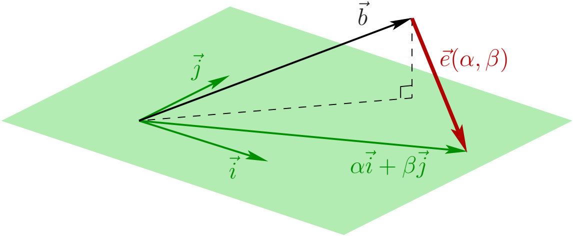
Solve for $\arg\min\limits_{\alpha,\beta} \|\vec{e}(\alpha,\beta)\|$, where $\vec{e}(\alpha,\beta) := \alpha\vec{i} + \beta\vec{j} - \vec{b}$:



Back to the linear regression



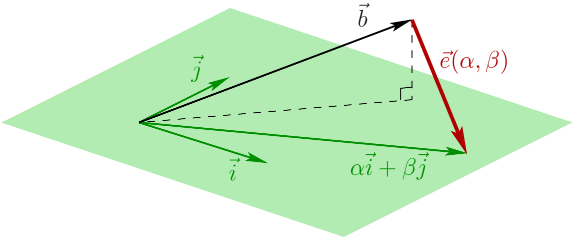
Back to the linear regression



The $\|\vec{e}(\alpha, \beta)\|$ is minimized when $\vec{e}(\alpha, \beta) \perp \operatorname{span}\{\vec{i}, \vec{j}\}$:

$$\begin{cases} \vec{i}^\top \vec{e}(\alpha, \beta) = 0 \\ \vec{j}^\top \vec{e}(\alpha, \beta) = 0 \end{cases}$$

Back to the linear regression

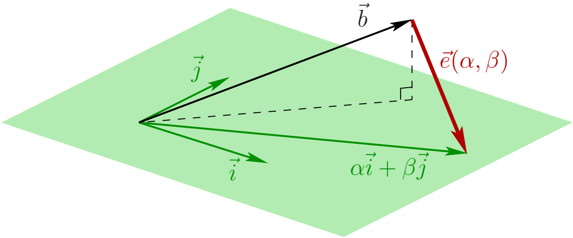


The $\|\vec{e}(\alpha,\beta)\|$ is minimized when $\vec{e}(\alpha,\beta) \perp \text{span}\{\vec{i},\vec{j}\}$:

$$\begin{cases} \vec{i}^\top \vec{e}(\alpha,\beta) = 0 \\ \vec{j}^\top \vec{e}(\alpha,\beta) = 0 \end{cases}$$

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{bmatrix} \left(\alpha \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Back to the linear regression



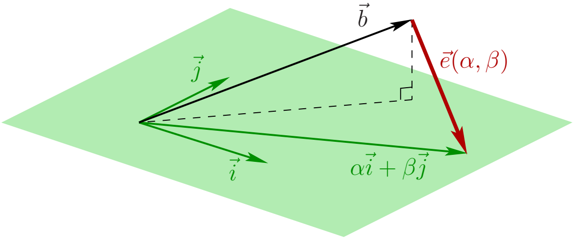
The $\|\vec{e}(\alpha, \beta)\|$ is minimized when $\vec{e}(\alpha, \beta) \perp \text{span}\{\vec{i}, \vec{j}\}$:

$$\begin{cases} \vec{i}^\top \vec{e}(\alpha, \beta) = 0 \\ \vec{j}^\top \vec{e}(\alpha, \beta) = 0 \end{cases}$$

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{bmatrix} \left(\alpha \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$A^\top (Ax - b) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Back to the linear regression



The $\|\vec{e}(\alpha,\beta)\|$ is minimized when $\vec{e}(\alpha,\beta) \perp \text{span}\{\vec{i},\vec{j}\}$:

$$\begin{cases} \vec{i}^\top \vec{e}(\alpha,\beta) = 0 \\ \vec{j}^\top \vec{e}(\alpha,\beta) = 0 \end{cases}$$

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{bmatrix} \left(\alpha \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad A^\top(Ax - b) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

In a general case the matrix $A^\top A$ can be invertible!

$$A^\top A x = A^\top b.$$

Some nice properties of $A^\top A$

Theorem

$A^\top A$ is symmetric.

Some nice properties of $A^\top A$

Theorem

$A^\top A$ is symmetric.

It is very easy to show:

$$(A^\top A)^\top = A^\top (A^\top)^\top = A^\top A.$$

Some nice properties of $A^\top A$

Theorem

$A^\top A$ is symmetric.

It is very easy to show:

$$(A^\top A)^\top = A^\top (A^\top)^\top = A^\top A.$$

Theorem

$A^\top A$ positive semidefinite: $\forall x \in \mathbb{R}^n \quad x^\top A^\top A x \geq 0.$

Some nice properties of $A^\top A$

Theorem

$A^\top A$ is symmetric.

It is very easy to show:

$$(A^\top A)^\top = A^\top (A^\top)^\top = A^\top A.$$

Theorem

$A^\top A$ positive semidefinite: $\forall x \in \mathbb{R}^n \quad x^\top A^\top A x \geq 0.$

It follows from the fact that $x^\top A^\top A x = (Ax)^\top A x > 0$. Moreover, $A^\top A$ is positive definite in the case where A has linearly independent columns (rank A is equal to the number of the variables in the system).

Least squares in more than two dimensions

The same reasoning applies, here is an algebraic way to show it:

$$\arg\min \|Ax - b\|^2$$

Least squares in more than two dimensions

The same reasoning applies, here is an algebraic way to show it:

$$\arg\min \|Ax - b\|^2 = \arg\min (Ax - b)^\top (Ax - b) =$$

Least squares in more than two dimensions

The same reasoning applies, here is an algebraic way to show it:

$$\arg\min \|Ax - b\|^2 = \arg\min (Ax-b)^\top (Ax-b) = \arg\min (x^\top A^\top - b^\top)(Ax - b) =$$

Least squares in more than two dimensions

The same reasoning applies, here is an algebraic way to show it:

$$\begin{aligned}\arg\min \|Ax - b\|^2 = \arg\min (Ax - b)^\top (Ax - b) = \arg\min (x^\top A^\top - b^\top)(Ax - b) = \\ = \arg\min (x^\top A^\top A x - b^\top A x - x^\top A^\top b + \underbrace{b^\top b}_{\text{const}}) =\end{aligned}$$

Least squares in more than two dimensions

The same reasoning applies, here is an algebraic way to show it:

$$\begin{aligned}\arg\min \|Ax-b\|^2 &= \arg\min (Ax-b)^\top(Ax-b) = \arg\min (x^\top A^\top - b^\top)(Ax-b) = \\ &= \arg\min (x^\top A^\top Ax - b^\top Ax - x^\top A^\top b + \underbrace{b^\top b}_{\text{const}}) = \\ &= \arg\min (x^\top A^\top Ax - 2b^\top Ax) =\end{aligned}$$

Least squares in more than two dimensions

The same reasoning applies, here is an algebraic way to show it:

$$\begin{aligned}\arg\min \|Ax - b\|^2 &= \arg\min (Ax-b)^\top (Ax-b) = \arg\min (x^\top A^\top - b^\top)(Ax-b) = \\ &= \arg\min (x^\top A^\top A x - b^\top A x - x^\top A^\top b + \underbrace{b^\top b}_{\text{const}}) = \\ &= \arg\min (x^\top A^\top A x - 2 b^\top A x) = \arg\min (x^\top \underbrace{(A^\top A)}_{:=A'} x - 2 \underbrace{(A^\top b)}_{:=b'}{}^\top x)\end{aligned}$$

Least squares in more than two dimensions

The same reasoning applies, here is an algebraic way to show it:

$$\begin{aligned}\arg\min \|Ax - b\|^2 &= \arg\min (Ax-b)^\top (Ax-b) = \arg\min (x^\top A^\top - b^\top)(Ax-b) = \\ &= \arg\min (x^\top A^\top A x - b^\top A x - x^\top A^\top b + \underbrace{b^\top b}_{\text{const}}) = \\ &= \arg\min (x^\top A^\top A x - 2 b^\top A x) = \arg\min (x^\top \underbrace{(A^\top A)}_{:=A'} x - 2 \underbrace{(A^\top b)}_{:=b'}{}^\top x)\end{aligned}$$

The takeaway message

The least squares problem $\arg\min \|Ax - b\|^2$ is equivalent to minimizing the quadratic function $\arg\min \left(x^\top A' x - 2b'^\top x\right)$ with (in general) a symmetric positive definite matrix A'. This can be done by solving a linear system $A'x = b'$.

Table of Contents

Linear-quadratic regulator

Imagine a car going at $v_0 = 0.5$ m/s. The goal is to accelerate to $v_n = 2.3$ m/s in $n = 30$ s maximum. We can control the acceleration u_i via the gas pedal:

$$v_{i+1} = v_i + u_i$$

Linear-quadratic regulator

Imagine a car going at $v_0 = 0.5$ m/s. The goal is to accelerate to $v_n = 2.3$ m/s in $n = 30$ s maximum. We can control the acceleration u_i via the gas pedal:

$$v_{i+1} = v_i + u_i$$

So, we need to find $\{u_i\}_{i=0}^{n-1}$ that optimizes some quality criterion $J(\vec{v}, \vec{u})$:

$$\arg\min J(\vec{v}, \vec{u}) \quad s.t. \quad v_{i+1} = v_i + u_i = v_0 + \sum_{j=0}^{i-1} u_j \quad \forall i \in 0..n-1$$

Linear-quadratic regulator

Imagine a car going at $v_0 = 0.5$ m/s. The goal is to accelerate to $v_n = 2.3$ m/s in $n = 30$ s maximum. We can control the acceleration u_i via the gas pedal:

$$v_{i+1} = v_i + u_i$$

So, we need to find $\{u_i\}_{i=0}^{n-1}$ that optimizes some quality criterion $J(\vec{v}, \vec{u})$:

$$\arg\min J(\vec{v}, \vec{u}) \quad s.t. \quad v_{i+1} = v_i + u_i = v_0 + \sum_{j=0}^{i-1} u_j \quad \forall i \in 0..n-1$$

What happens if we ask for the car to reach the final speed as quickly as possible?
It can be written as follows:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} (v_i - v_n)^2 = \sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2$$

Linear-quadratic regulator

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2$$

Solve in the least squares sense:

$$\begin{cases} u_0 & & & & = v_n - v_0 \\ u_0 & +u_1 & & & = v_n - v_0 \\ \vdots & & \ddots & & \vdots \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \end{cases}$$

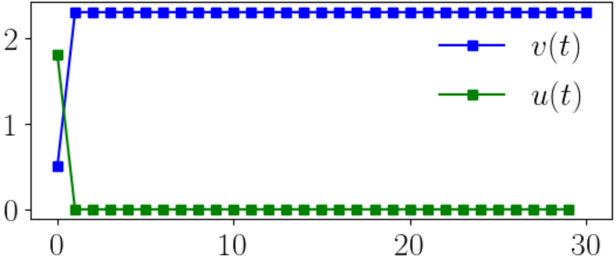
Linear-quadratic regulator

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2$$

Solve in the least squares sense:

$$\begin{cases} u_0 & & & & = v_n - v_0 \\ u_0 & +u_1 & & & = v_n - v_0 \\ \vdots & & \ddots & & \vdots \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \end{cases}$$

Ouch... Quite brutal accelerations: obvious solution $u_0 = v_n - v_0$, $u_i = 0\ \forall i > 0$.



Linear-quadratic regulator

Ok, no problem, let us penalize large accelerations:

$$J(\vec{v}, \vec{u}) := \sum_{i=0}^{n-1} {u_i}^2 + \left(\sum_{i=0}^{n-1} u_i - v_n\right)^2$$

Solve in the least squares sense:

$$\begin{cases} u_0 & & & & = 0 \\ & u_1 & & & = 0 \\ & & \ddots & & \vdots \\ & & & u_{n-1} & = 0 \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \end{cases}$$

Linear-quadratic regulator

Ok, no problem, let us penalize large accelerations:

$$J(\vec{v}, \vec{u}) := \sum_{i=0}^{n-1} {u_i}^2 + \left(\sum_{i=0}^{n-1} u_i - v_n\right)^2$$

Solve in the least squares sense:

$$\begin{cases} u_0 & & & & = 0 \\ & u_1 & & & = 0 \\ & & \ddots & & \vdots \\ & & & u_{n-1} & = 0 \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \end{cases}$$

```
import numpy as np
n,v0,vn = 30,0.5,2.3
A = np.matrix(np.vstack((np.diag([1]*n), [1]*n)))
b = np.matrix([[0]]*n + [[vn-v0]])
u = np.linalg.inv(A.T*A)*A.T*b
v = [v0 + np.sum(u[:i]) for i in range(0,n+1)]
```

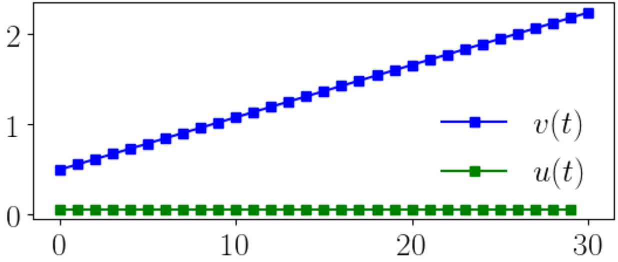
Linear-quadratic regulator

Ok, no problem, let us penalize large accelerations:

$$J(\vec{v},\vec{u}) := \sum_{i=0}^{n-1} {u_i}^2 + \left(\sum_{i=0}^{n-1} u_i - v_n\right)^2$$

Solve in the least squares sense:

$$\begin{cases} u_0 & & & & = 0 \\ & u_1 & & & = 0 \\ & & \ddots & & \vdots \\ & & & u_{n-1} & = 0 \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \end{cases}$$



Low acceleration, however the transient time becomes unacceptable.

Linear-quadratic regulator

Optimize for competing goals:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} (v_i - v_n)^2 + \boxed{4} \sum_{i=0}^{n-1} u_i^2$$

Linear-quadratic regulator

Optimize for competing goals:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n}(v_i - v_n)^2 + \boxed{4}\sum_{i=0}^{n-1} {u_i}^2 =$$

$$\sum_{i=1}^{n}\left(\sum_{j=0}^{i-1} u_j - v_n + v_0\right)^2 + \boxed{4}\sum_{i=0}^{n-1} {u_i}^2$$

N.B. Note the tradeoff coefficients !

Linear-quadratic regulator

Optimize for competing goals:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} (v_i - v_n)^2 + \boxed{4} \sum_{i=0}^{n-1} u_i^2 =$$

$$\sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2 + \boxed{4} \sum_{i=0}^{n-1} u_i^2$$

N.B. Note the tradeoff coefficients!

$$\begin{cases}
u_0 & & & & = v_n - v_0 \\
u_0 & +u_1 & & & = v_n - v_0 \\
\vdots & & \ddots & & \vdots \\
u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \\
\boxed{2}\, u_0 & & & & = 0 \\
& \boxed{2}\, u_1 & & & = 0 \\
& & \ddots & & \vdots \\
& & & \boxed{2}\, u_{n-1} & = 0
\end{cases}$$

Linear-quadratic regulator

Optimize for competing goals:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} (v_i - v_n)^2 + \boxed{4} \sum_{i=0}^{n-1} u_i^2 =$$

$$\sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2 + \boxed{4} \sum_{i=0}^{n-1} u_i^2$$

N.B. Note the tradeoff coefficients!

$$\begin{cases} u_0 & & & & = v_n - v_0 \\ u_0 & +u_1 & & & = v_n - v_0 \\ \vdots & & \ddots & & \vdots \\ u_0 & +u_1 & \dots & +u_{n-1} & = v_n - v_0 \\ \boxed{2}\, u_0 & & & & = 0 \\ & \boxed{2}\, u_1 & & & = 0 \\ & & \ddots & & \vdots \\ & & & \boxed{2}\, u_{n-1} & = 0 \end{cases}$$

```python
import numpy as np
n,v0,vn = 30,0.5,2.3
A = np.matrix(np.vstack((np.tril(np.ones((n,n))), np.diag([2]*n))))
b = np.matrix([[vn-v0]]*n + [[0]]*n)
u = np.linalg.inv(A.T*A)*A.T*b
v = [v0 + np.sum(u[:i]) for i in range(0,n+1)]
```

Linear-quadratic regulator

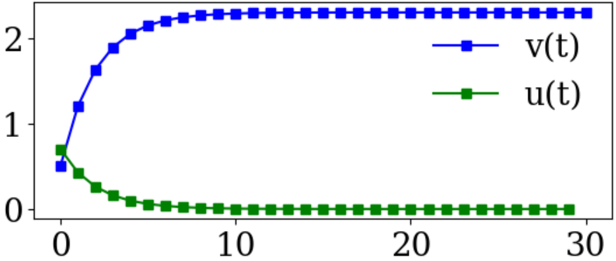
Optimize for competing goals:

$$J(\vec{v}, \vec{u}) := \sum_{i=1}^{n} (v_i - v_n)^2 + \boxed{4} \sum_{i=0}^{n-1} {u_i}^2 =$$

$$\sum_{i=1}^{n} \left(\sum_{j=0}^{i-1} u_j - v_n + v_0 \right)^2 + \boxed{4} \sum_{i=0}^{n-1} {u_i}^2$$

N.B. Note the tradeoff coefficients!

$$\left\{\begin{array}{llll} u_0 & & & = v_n - v_0 \\ u_0 & +u_1 & & = v_n - v_0 \\ \vdots & & \ddots & \vdots \\ u_0 & +u_1 & \dots \quad +u_{n-1} & = v_n - v_0 \\ \boxed{2}\, u_0 & & & = 0 \\ & \boxed{2}\, u_1 & & = 0 \\ & & \ddots & \vdots \\ & & \boxed{2}\, u_{n-1} & = 0 \end{array}\right.$$



Linear-quadratic regulator

The takeaway message

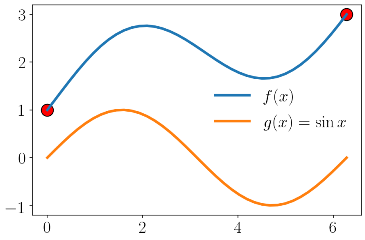
For the same problem, the same choice of variables, tweaking the objective function produces very different results. Use it!

Poisson's equation

Problem: find $f(x)$ defined on $x \in [0, 2\pi]$ as close as possible to $g(x) := \sin x$, constrained to $f(0) = 1$ and $f(2\pi) = 3$.

Formulate it as the Poisson's equation with Dirichlet boundary conditions:

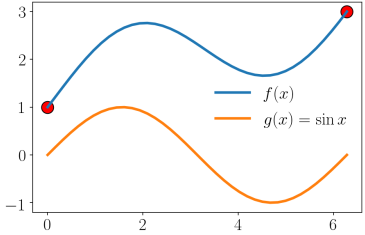
$\frac{d^2}{dx^2} f = \frac{d^2}{dx^2} g \quad$ s.t. $f(0) = 1, \; f(2\pi) = 3$



Poisson's equation

Problem: find $f(x)$ defined on $x \in [0, 2\pi]$ as close as possible to $g(x) := \sin x$, constrained to $f(0) = 1$ and $f(2\pi) = 3$.

Formulate it as the Poisson's equation with Dirichlet boundary conditions:
$\frac{d^2}{dx^2} f = \frac{d^2}{dx^2} g \quad \text{s.t. } f(0) = 1,\ f(2\pi) = 3$



Neanderthal method:

```
import numpy as np
n,f0,fn = 32,1.,3.
g = [np.sin(x) for x in np.linspace(0, 2*np.pi, n)]
f = [f0] + [0]*(n-2) + [fn]
for _ in range(512):
    for i in range(1, n-1):
        f[i] = ( f[i-1] + f[i+1] + (2*g[i]-g[i-1]-g[i+1]) )/2.
```

N.B: extremely slow convergence for larger problems, very hard to build upon

Poisson's equation

Least squares formulation:

$$\arg\min_{f} \int_0^{2\pi} \| f' - g' \|^2$$

with $f(0) = 1, \ f(2\pi) = 3$

Poisson's equation

Least squares formulation:

$$\arg\min_{f} \int_0^{2\pi} \|f' - g'\|^2$$

with $f(0) = 1,\ f(2\pi) = 3$

Discretization:

$$\left\{\begin{array}{rrrrl} f_1 & & & & = g_1 - g_0 + f_0 \\ -f_1 & +f_2 & & & = g_2 - g_1 \\ & \ddots & \ddots & & \vdots \\ & & -f_{n-3} & +f_{n-2} & = g_{n-2} - g_{n-3} \\ & & & -f_{n-2} & = g_{n-1} - g_{n-2} - f_{n-1} \end{array}\right.$$

Poisson's equation

Least squares formulation:

$$\arg\min_{f} \int_0^{2\pi} \|f' - g'\|^2$$

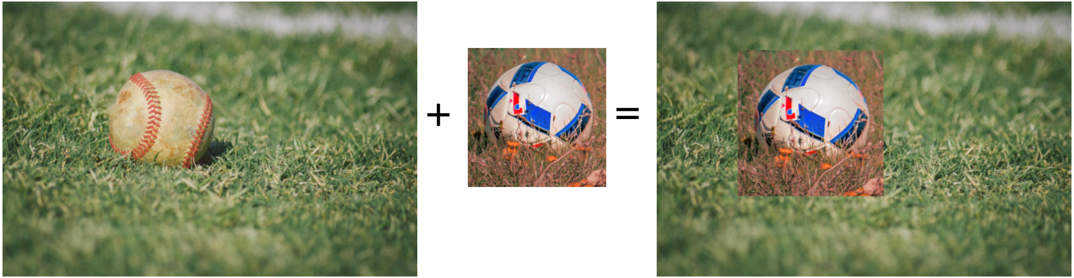
with $f(0) = 1, \; f(2\pi) = 3$

Discretization:

$$\begin{cases} f_1 & & & & = g_1 - g_0 + f_0 \\ -f_1 & +f_2 & & & = g_2 - g_1 \\ & \ddots & \ddots & & \vdots \\ & & -f_{n-3} & +f_{n-2} & = g_{n-2} - g_{n-3} \\ & & & -f_{n-2} & = g_{n-1} - g_{n-2} - f_{n-1} \end{cases}$$

```
import numpy as np
n,f0,fn = 32,1.,3.
g = [np.sin(x) for x in np.linspace(0, 2*np.pi, n)]
A = np.matrix(np.zeros((n-1,n-2)))
np.fill_diagonal(A,       1)
np.fill_diagonal(A[1:], -1)
b = np.matrix([[g[i]-g[i-1]] for i in range(1,n)])
b[ 0,0] = b[ 0,0] + f0
b[-1,0] = b[-1,0] - fn
f = [f0] + (np.linalg.inv(A.T*A)*A.T*b).T.tolist()[0] + [fn]
```

Poisson image editing



Poisson image editing

 +  = 

We can do better: solve a linear system per color channel.

Poisson image editing


+

=


We can do better: solve a linear system per color channel.

Let a be:

$a : \Omega \to \mathbb{R}$

Poisson image editing

 +  = 

We can do better: solve a linear system per color channel.

Let a be:

$a : \Omega \to \mathbb{R}$

Let b be:

$b : \Omega \to \mathbb{R}$

Poisson image editing

 +  = 

We can do better: solve a linear system per color channel.

Let a be:

$a : \Omega \to \mathbb{R}$

Let b be:

$b : \Omega \to \mathbb{R}$

Solve for f who takes its boundary conditions from a and the gradients from b:

$$\arg\min_{f} \int_{\Omega} \|\nabla f - \nabla b\|^2 \quad \text{with } f|_{\partial\Omega} = a|_{\partial\Omega}$$

Poisson image editing

Discretize the problem: having $w \times h$ pixels grayscale images a and b, we compute a $w \times h$ pixels image f, solve in the least squares sense:

$$\begin{cases} f_{i+1,j} - f_{i,j} &= b_{i+1,j} - b_{i,j} \quad \forall (i,j) \in [0 \dots w-2] \times [0 \dots h-2] \\ f_{i,j+1} - f_{i,j} &= b_{i,j+1} - b_{i,j} \quad \forall (i,j) \in [0 \dots w-2] \times [0 \dots h-2] \\ f_{i,j} &= a_{i,j} \quad \forall (i,j) \text{ s.t. } i=0 \text{ or } i=w-1 \text{ or } j=0 \text{ or } j=h-1 \end{cases}$$

N.B: sparse system solver!

Poisson image editing

Discretize the problem: having $w \times h$ pixels grayscale images a and b, we compute a $w \times h$ pixels image f, solve in the least squares sense:

$$\begin{cases} f_{i+1,j} - f_{i,j} &= b_{i+1,j} - b_{i,j} \quad \forall (i,j) \in [0 \dots w-2] \times [0 \dots h-2] \\ f_{i,j+1} - f_{i,j} &= b_{i,j+1} - b_{i,j} \quad \forall (i,j) \in [0 \dots w-2] \times [0 \dots h-2] \\ f_{i,j} &= a_{i,j} \quad \forall (i,j) \text{ s.t. } i=0 \text{ or } i=w-1 \text{ or } j=0 \text{ or } j=h-1 \end{cases}$$

+

=

Poisson's equation

The takeaway message

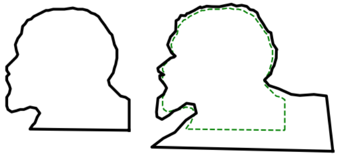
Poisson's problem is one of the most used tools in geometry processing.
Know your friends!

Caricature

```
x = [100,100,97,93 ...  23,21,19] # 2d closed polyline
y = [0,25,27,28,30 ...  11,6,4,1]
n = len(x)                        # number of points
cx = [x[i] - (x[(i-1+n)%n]+x[(i+1)%n])/2 for i in range(n)] #precompute the
cy = [y[i] - (y[(i-1+n)%n]+y[(i+1)%n])/2 for i in range(n)] #discrete curvature
for _ in range(1000): # Gauss-Seidel iterations
    for i in range(n):
        x[i] = (x[(i-1+n)%n]+x[(i+1)%n])/2 + cx[i]*1.9
        y[i] = (y[(i-1+n)%n]+y[(i+1)%n])/2 + cy[i]*1.9
```

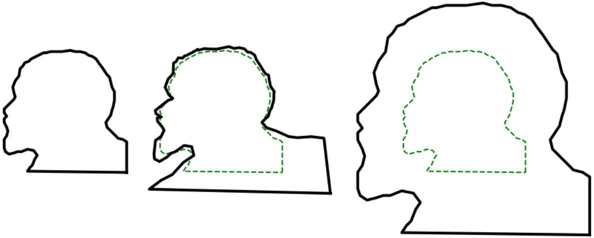
Caricature

```
x = [100,100,97,93 ...  23,21,19] # 2d closed polyline
y = [0,25,27,28,30 ...  11,6,4,1]
n = len(x)                        # number of points
cx = [x[i] - (x[(i-1+n)%n]+x[(i+1)%n])/2 for i in range(n)] #precompute the
cy = [y[i] - (y[(i-1+n)%n]+y[(i+1)%n])/2 for i in range(n)] #discrete curvature
for _ in range(1000): # Gauss-Seidel iterations
    for i in range(n):
        x[i] = (x[(i-1+n)%n]+x[(i+1)%n])/2 + cx[i]*1.9
        y[i] = (y[(i-1+n)%n]+y[(i+1)%n])/2 + cy[i]*1.9
```



Caricature

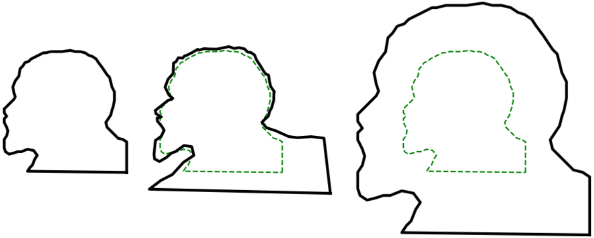
```
x = [100,100,97,93 ...  23,21,19] # 2d closed polyline
y = [0,25,27,28,30 ...  11,6,4,1]
n = len(x)                        # number of points
cx = [x[i] - (x[(i-1+n)%n]+x[(i+1)%n])/2 for i in range(n)] #precompute the
cy = [y[i] - (y[(i-1+n)%n]+y[(i+1)%n])/2 for i in range(n)] #discrete curvature
for _ in range(1000): # Gauss-Seidel iterations
    for i in range(n):
        x[i] = (x[(i-1+n)%n]+x[(i+1)%n])/2 + cx[i]*1.9
        y[i] = (y[(i-1+n)%n]+y[(i+1)%n])/2 + cy[i]*1.9
```



Almost, but no :(

Caricature

```
x = [100,100,97,93 ...  23,21,19] # 2d closed polyline
y = [0,25,27,28,30 ...  11,6,4,1]
n = len(x)                        # number of points
cx = [x[i] - (x[(i-1+n)%n]+x[(i+1)%n])/2 for i in range(n)] #precompute the
cy = [y[i] - (y[(i-1+n)%n]+y[(i+1)%n])/2 for i in range(n)] #discrete curvature
for _ in range(1000): # Gauss-Seidel iterations
    for i in range(n):
        x[i] = (x[(i-1+n)%n]+x[(i+1)%n])/2 + cx[i]*1.9
        y[i] = (y[(i-1+n)%n]+y[(i+1)%n])/2 + cy[i]*1.9
```



Almost, but no :(

Least squares equivalent:

$$\underset{\{x'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } (i,j)} \left(x'_j - x'_i - c \cdot (x_j - x_i) \right)^2$$

x_i are the input coordinates and x'_i are the unknowns (separable in x and y)

Caricature

A quick fix:

$$\arg\min_{\{x'_i\}_{i=0}^{n-1}} \sum_{\forall \text{ edge } (i,j)} \left(x'_j - x'_i - c_0 \cdot (x_j - x_i) \right)^2$$

Caricature

A quick fix:

$$\arg\min_{\{x'_i\}_{i=0}^{n-1}} \sum_{\forall \text{ edge } (i,j)} \left(x'_j - x'_i - c_0 \cdot (x_j - x_i) \right)^2 \boxed{+ \sum_{\forall \text{ vertex } i} c_1^2 \left(x'_i - x_i\right)^2}$$

Caricature

A quick fix:

$$\arg\min_{\{x_i'\}_{i=0}^{n-1}} \sum_{\forall \text{ edge } (i,j)} \left(x_j' - x_i' - c_0 \cdot (x_j - x_i)\right)^2 + \sum_{\forall \text{ vertex } i} c_1^2 \left(x_i' - x_i\right)^2$$

$$\begin{cases}
\begin{array}{cccccc}
-x_0' & +x_1' & & & & = c_0 \cdot (x_1 - x_0) \\
 & -x_1' & +x_2' & & & = c_0 \cdot (x_2 - x_1) \\
 & & \ddots & \ddots & & \vdots \\
 & & & -x_{n-2}' & +x_{n-1}' & = c_0 \cdot (x_{n-2} - x_{n-1}) \\
x_0' & & & & -x_{n-1}' & = c_0 \cdot (x_{n-1} - x_0) \\
c_1 \cdot x_0' & & & & & = c_1 \cdot x_0 \\
 & c_1 \cdot x_1' & & & & = c_1 \cdot x_1 \\
 & & & \ddots & & \vdots \\
 & & & & c_1 \cdot x_{n-1}' & = c_1 \cdot x_{n-1}
\end{array}
\end{cases}$$

Caricature

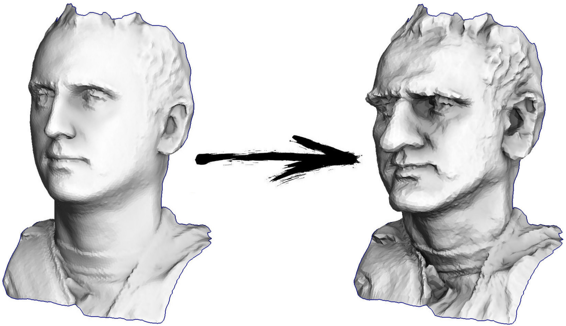
A quick fix:

$$\underset{\{x'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } (i,j)} \left(x'_j - x'_i - c_0 \cdot (x_j - x_i)\right)^2 \boxed{+ \sum_{\forall \text{ vertex } i} c_1^2 \left(x'_i - x_i\right)^2}$$

Caricature

A quick fix:

$$\underset{\{x'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } (i,j)} \left(x'_j - x'_i - c_0 \cdot (x_j - x_i)\right)^2 + \sum_{\forall \text{ vertex } i} c_1^2 \left(x'_i - x_i\right)^2$$



And it works out of the box for 3d surfaces as well!

Caricature

A quick fix:

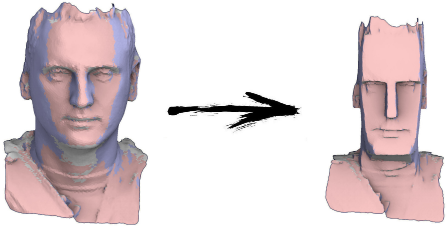
$$\underset{\{x'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } (i,j)} \left(x'_j - x'_i - c_0 \cdot (x_j - x_i)\right)^2 + \sum_{\forall \text{ vertex } i} c_1^2 \left(x'_i - x_i\right)^2$$

The takeaway message

Reformulating as a least squares problem allows for much easier tweaking.

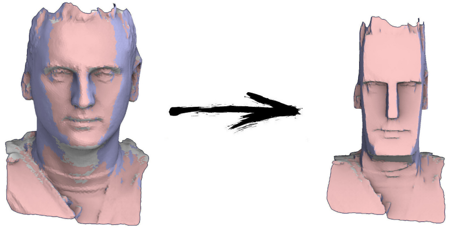
Cubify it!

$$\vec{a}_{ijk} := \underset{\vec{a}\in\{(1,0,0),(0,1,0),(0,0,1)\}}{\arg\max} \left|\vec{a}\cdot\vec{N}_{ijk}\right|$$



Cubify it!

$$\vec{a}_{ijk} := \underset{\vec{a}\in\{(1,0,0),(0,1,0),(0,0,1)\}}{\arg\max} \left|\vec{a}\cdot\vec{N}_{ijk}\right|$$



Let $\vec{e}_{ij} := \vec{x}_j - \vec{x}_i$ be the input geometry, and $\vec{e}'_{ij} := \vec{x}'_j - \vec{x}'_i$ the unknowns.

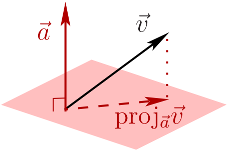
Quick test

What would be the result?

$$\underset{\{\vec{x}'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\|\vec{e}'_{ij} - \vec{e}_{ij}\right\|^2$$

Cubify it!

$$\vec{a}_{ijk} := \underset{\vec{a}\in\{(1,0,0),(0,1,0),(0,0,1)\}}{\arg\max} \left|\vec{a}\cdot\vec{N}_{ijk}\right|$$



$$\mathrm{proj}_{\vec{a}}\vec{v} := \vec{v} - \frac{\vec{v}\cdot\vec{v}}{\vec{a}\cdot\vec{a}}\vec{a}$$

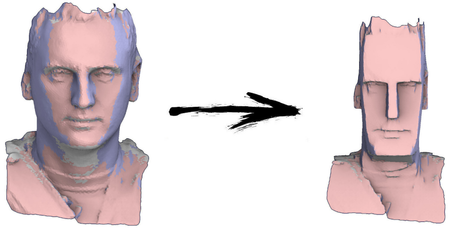
Let $\vec{e}_{ij} := \vec{x}_j - \vec{x}_i$ be the input geometry, and $\vec{e}'_{ij} := \vec{x}'_j - \vec{x}'_i$ the unknowns.

$$\underset{\{\vec{x}'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\|\vec{e}'_{ij} - \vec{e}_{ij}\right\|^2 + \sum_{\forall \text{ triangle } ijk} \boxed{c} \cdot \left(\left\|\vec{e}'_{ij} - \mathrm{proj}_{\vec{a}_{ijk}} \vec{e}_{ij}\right\|^2 + \left\|\vec{e}'_{jk} - \mathrm{proj}_{\vec{a}_{ijk}} \vec{e}_{jk}\right\|^2 + \left\|\vec{e}'_{ki} - \mathrm{proj}_{\vec{a}_{ijk}} \vec{e}_{ki}\right\|^2 \right)$$

N.B: still a separable problem

Cubify it!

$$\vec{a}_{ijk} := \underset{\vec{a}\in\{(1,0,0),(0,1,0),(0,0,1)\}}{\arg\max} \left|\vec{a}\cdot\vec{N}_{ijk}\right|$$



Let $\vec{e}_{ij} := \vec{x}_j - \vec{x}_i$ be the input geometry, and $\vec{e}'_{ij} := \vec{x}'_j - \vec{x}'_i$ the unknowns.

$$\underset{\{\vec{x}'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall\text{ edge } ij} \left\|\vec{e}'_{ij} - \vec{e}_{ij}\right\|^2 + \sum_{\forall\text{ triangle } ijk} \boxed{c}\cdot\left(\left\|\vec{e}'_{ij} - \text{proj}_{\vec{a}_{ijk}}\vec{e}_{ij}\right\|^2 + \left\|\vec{e}'_{jk} - \text{proj}_{\vec{a}_{ijk}}\vec{e}_{jk}\right\|^2 + \left\|\vec{e}'_{ki} - \text{proj}_{\vec{a}_{ijk}}\vec{e}_{ki}\right\|^2\right)$$

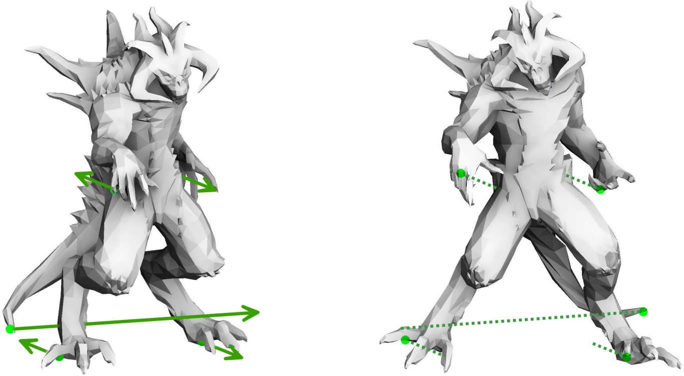
The takeaway message

And again, the same variables, but different tweaking
$\Rightarrow$ completely different results.

N.B: still a separable problem

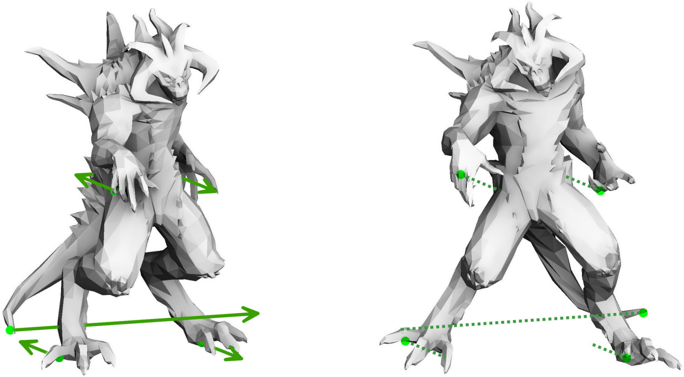
As-rigid-as-possible deformation

Problem: compute a deformation of a mesh with several constrained vertex positions.



As-rigid-as-possible deformation

Problem: compute a deformation of a mesh with several constrained vertex positions.



Let $\vec{e}_{ij} := \vec{x}_j - \vec{x}_i$ be the input geometry, and $\vec{e}\,'_{ij} := \vec{x}\,'_j - \vec{x}\,'_i$ the unknowns.
Choose a subset of vertices $\mathcal{I} \subset [0 \dots n-1]$, to have final position $\{\vec{p}_k\}_{k\in\mathcal{I}}$

As-rigid-as-possible deformation

Let $\vec{e}_{ij} := \vec{x}_j - \vec{x}_i$ be the input geometry, and $\vec{e}'_{ij} := \vec{x}'_j - \vec{x}'_i$ the unknowns.
Choose a subset of vertices $\mathcal{I} \subset [0 \dots n-1]$, to have final position $\{\vec{p}_k\}_{k \in \mathcal{I}}$

Naive solution

$$\underset{\{\vec{x}'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}'_{ij} - \vec{e}_{ij} \right\|^2 \text{ subject to the constraints } \vec{x}'_k = p_k \ \ \forall k \in \mathcal{I}$$

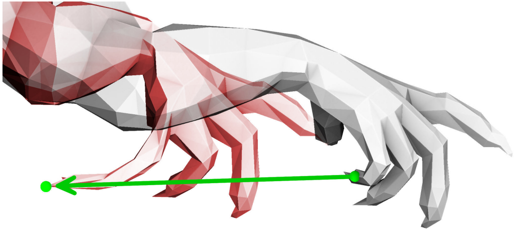
As-rigid-as-possible deformation

Let $\vec{e}_{ij} := \vec{x}_j - \vec{x}_i$ be the input geometry, and $\vec{e}'_{ij} := \vec{x}'_j - \vec{x}'_i$ the unknowns.
Choose a subset of vertices $\mathcal{I} \subset [0 \dots n-1]$, to have final position $\{\vec{p}_k\}_{k \in \mathcal{I}}$

Naive solution

$$\underset{\{\vec{x}'_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}'_{ij} - \vec{e}_{ij} \right\|^2 \text{ subject to the constraints } \vec{x}'_k = p_k \;\; \forall k \in \mathcal{I}$$

Problem: huge stretching near the constraints



As-rigid-as-possible deformation

Penalize stretching: make the deformation be a rotation locally

Introduce new variables: a rotation matrix R_i per vertex

$$\underset{\{\vec{x}_i', R_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}_{ij}' - R_i \times \vec{e}_{ij} \right\|^2$$

As-rigid-as-possible deformation

Penalize stretching: make the deformation be a rotation locally

Introduce new variables: a rotation matrix R_i per vertex

$$\underset{\{\vec{x}'_i, R_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}'_{ij} - R_i \times \vec{e}_{ij} \right\|^2$$

N.B: it is a non-linear problem!

As-rigid-as-possible deformation

Penalize stretching: make the deformation be a rotation locally

Introduce new variables: a rotation matrix R_i per vertex

$$\mathop{\arg\min}_{\{\vec{x}_i', R_i\}_{i=0}^{n-1}} \sum_{\forall \text{ edge } ij} \left\| \vec{e}_{ij}' - R_i \times \vec{e}_{ij} \right\|^2$$

N.B: it is a non-linear problem!

Solve alternatively for the vertex positions $\{\vec{x}_i'\}_{i=0}^{n-1}$ and rotations $\{R_i\}_{i=0}^{n-1}$:

As-rigid-as-possible deformation

Penalize stretching: make the deformation be a rotation locally

Introduce new variables: a rotation matrix R_i per vertex

$$\underset{\{\vec{x}_i', R_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}_{ij}' - R_i \times \vec{e}_{ij} \right\|^2$$

N.B: it is a non-linear problem!

Solve alternatively for the vertex positions $\{\vec{x}_i'\}_{i=0}^{n-1}$ and rotations $\{R_i\}_{i=0}^{n-1}$:

- Solving for $\{\vec{x}_i'\}_{i=0}^{n-1}$ having $\{R_i\}_{i=0}^{n-1}$ fixed is a **separable** linear problem: 3 conjugate gradients calls

As-rigid-as-possible deformation

Penalize stretching: make the deformation be a rotation locally

Introduce new variables: a rotation matrix R_i per vertex

$$\underset{\{\vec{x}_i', R_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}_{ij}' - R_i \times \vec{e}_{ij} \right\|^2$$

N.B: it is a non-linear problem!

Solve alternatively for the vertex positions $\{\vec{x}_i'\}_{i=0}^{n-1}$ and rotations $\{R_i\}_{i=0}^{n-1}$:

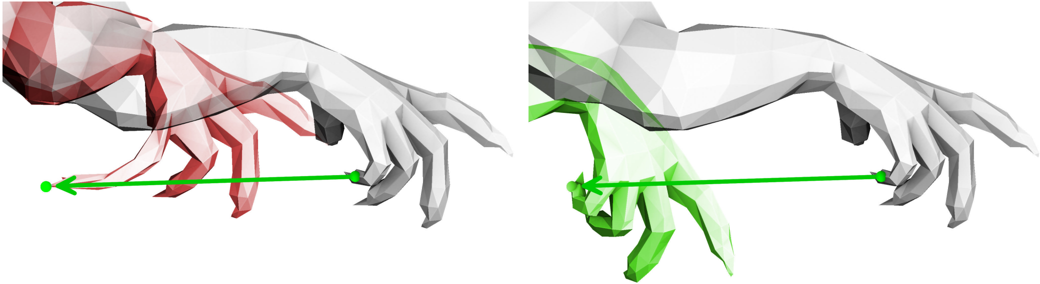
- Solving for $\{\vec{x}_i'\}_{i=0}^{n-1}$ having $\{R_i\}_{i=0}^{n-1}$ fixed is a **separable** linear problem: 3 conjugate gradients calls
- Solving for $\{R_i\}_{i=0}^{n-1}$ is the *orthogonal Procrustes problem* (closed form solution): let $U_i \Sigma_i V_i^\top$ be the s.v.d. of the 3×3 matrix $\sum_{j \text{ neighbor of } i} \vec{e}_{ij}' \vec{e}_{ij}^\top$, then $R_i \leftarrow U_i V_i^\top$

As-rigid-as-possible deformation

Penalize stretching: make the deformation be a rotation locally

Introduce new variables: a rotation matrix R_i per vertex

$$\underset{\{\vec{x}_i', R_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}_{ij}' - R_i \times \vec{e}_{ij} \right\|^2$$



As-rigid-as-possible deformation

Penalize stretching: make the deformation be a rotation locally

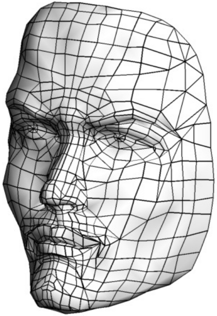
Introduce new variables: a rotation matrix R_i per vertex

$$\underset{\{\vec{x}'_i, R_i\}_{i=0}^{n-1}}{\arg\min} \sum_{\forall \text{ edge } ij} \left\| \vec{e}'_{ij} - R_i \times \vec{e}_{ij} \right\|^2$$

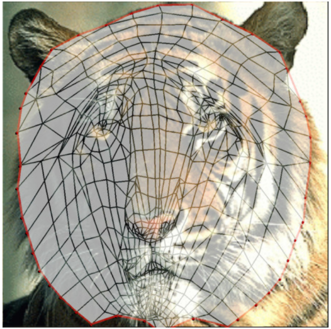
The takeaway message

Many nonlinear problems can be solved as a series of linear ones.

Mix the coordinates: least squares conformal maps

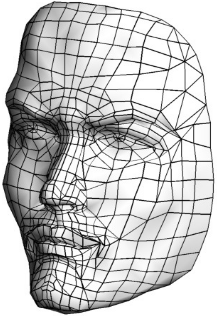


$$U(x, y, z) \longrightarrow = (u, v)$$

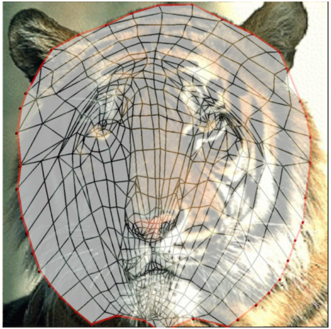


$$U^{-1}(u, v) \longrightarrow = (x, y, z)$$

Mix the coordinates: least squares conformal maps



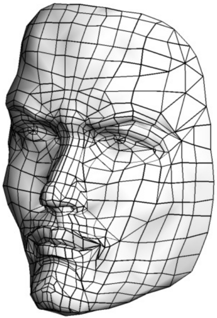
$$U(x, y, z) \longrightarrow = (u, v)$$



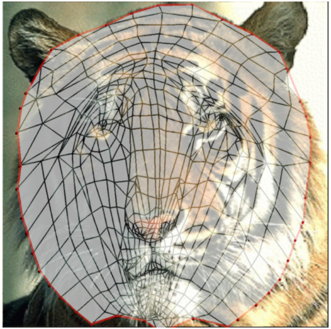
$$U^{-1}(u, v) \longrightarrow = (x, y, z)$$

How to create such a map?

Mix the coordinates: least squares conformal maps



$$U(x,y,z) = (u,v)$$



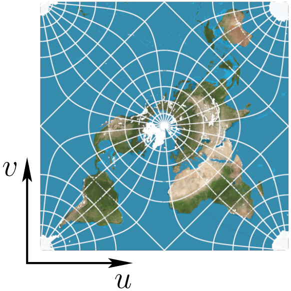
$$U^{-1}(u,v) = (x,y,z)$$

How to create such a map?

Let us compute a conformal map!

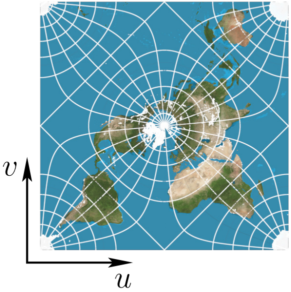
Mix the coordinates: least squares conformal maps

Maps that preserve angles
(but not distances or areas):



Mix the coordinates: least squares conformal maps

Maps that preserve angles
(but not distances or areas):

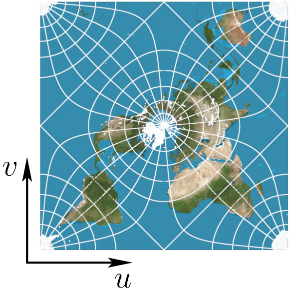


Cauchy–Riemann condition:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$
$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

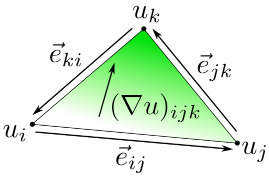
Mix the coordinates: least squares conformal maps

Maps that preserve angles
(but not distances or areas):



Cauchy–Riemann condition:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$
$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

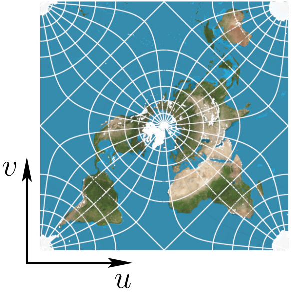


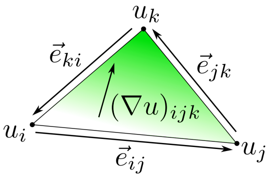
Sample tex coords at vertices: (u_i, v_i), interpolate linearly inside triangles.

$\Rightarrow$ The gradient is **constant** across each triangle

Mix the coordinates: least squares conformal maps

Maps that preserve angles (but not distances or areas):





Sample tex coords at vertices: (u_i, v_i), interpolate linearly inside triangles.
$\Rightarrow$ The gradient is **constant** across each triangle

Very simple formula for a gradient over a triangle:

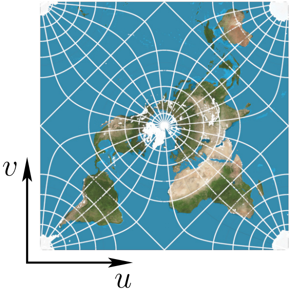
$$\vec{N}_{ijk} \times (\nabla u)_{ijk} = -\frac{1}{2A_{ijk}}(u_i \vec{e}_{jk} + u_j \vec{e}_{ki} + u_k \vec{e}_{ij})$$

Cauchy–Riemann condition:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$
$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

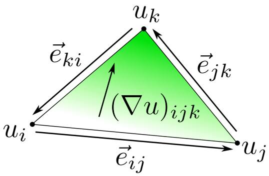
Mix the coordinates: least squares conformal maps

Maps that preserve angles
(but not distances or areas):



Cauchy–Riemann condition:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}$$
$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$



Sample tex coords at vertices: (u_i, v_i), interpolate linearly inside triangles.
$\Rightarrow$ The gradient is **constant** across each triangle

Very simple formula for a gradient over a triangle:

$$\vec{N}_{ijk} \times (\nabla u)_{ijk} = -\frac{1}{2A_{ijk}}(u_i \vec{e}_{jk} + u_j \vec{e}_{ki} + u_k \vec{e}_{ij})$$

Sum failure of Cauchy–Riemann condition to hold:

$$\arg\min_{u,v} \sum_{\forall \text{ triangle } ijk} A_{ijk} \left((\nabla u)_{ijk} - \vec{N}_{ijk} \times (\nabla v)_{ijk} \right)^2$$

N.B: Beware of the zero solution!

Mix the coordinates: least squares conformal maps

Quick hack: pin two arbitrary vertices.

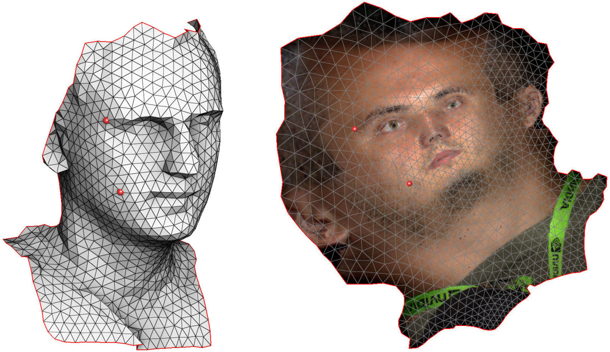


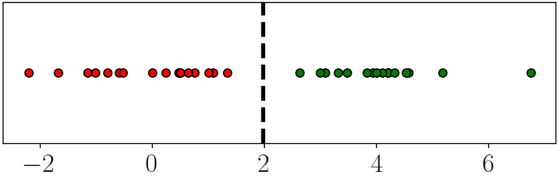
Table of Contents

Binary classification: a naive approach

Learning database: *n* real numbers and the corresponding colors (red and green).
Problem: build a classifying function $\mathbb{R} \to \{\text{red}, \text{green}\}$.

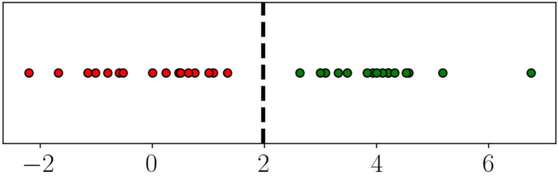
Binary classification: a naive approach

Learning database: *n* real numbers and the corresponding colors (red and green).
Problem: build a classifying function $\mathbb{R} \to \{\text{red}, \text{green}\}$.

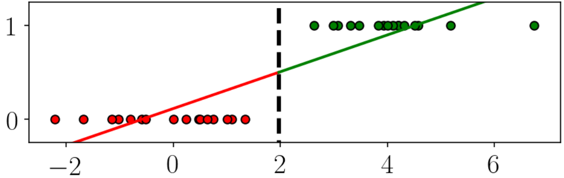


Binary classification: a naive approach

Learning database: n real numbers and the corresponding colors (red and green).
Problem: build a classifying function $\mathbb{R} \to \{\text{red}, \text{green}\}$.

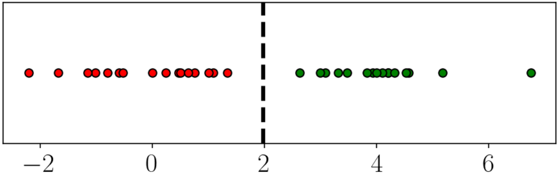


Encode red as 0 and green as 1, compute the regression line over $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathbb{R}$ and $y_i \in \{0, 1\}$. The decision rule: if $y(x) > 1/2$ then x is green, otherwise it is red.

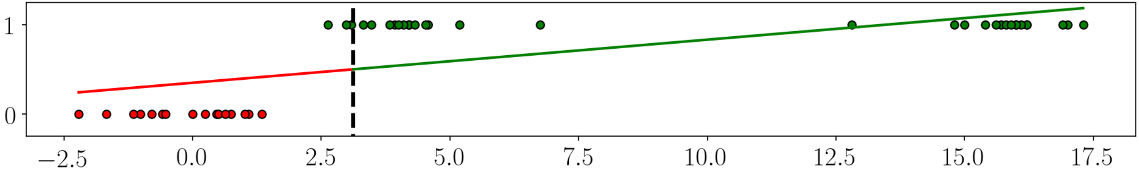


Binary classification: a naive approach

Learning database: n real numbers and the corresponding colors (red and green).
Problem: build a classifying function $\mathbb{R} \to \{\text{red}, \text{green}\}$.



Encode red as 0 and green as 1, compute the regression line over $\{(x_i, y_i)\}_{i=1}^{n}$, $x_i \in \mathbb{R}$ and $y_i \in \{0, 1\}$. The decision rule: if $y(x) > 1/2$ then x is green, otherwise it is red.



Logistic growth

Limited ressources model: a colony of the bacteria *B. dendroides* is growing in a Petri dish. The colony's area a can be modeled as a function of time t:

$$a(t) = \frac{c}{1 + e^{-wt - w_0}},$$

where c is the carrying capacity, w_0 is the initial population size and w is the growth rate.

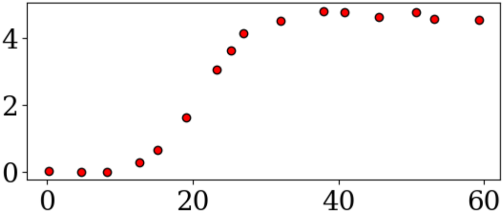
Logistic growth

Limited ressources model: a colony of the bacteria *B. dendroides* is growing in a Petri dish. The colony's area a can be modeled as a function of time t:

$$a(t) = \frac{c}{1 + e^{-wt - w_0}},$$

where c is the carrying capacity, w_0 is the initial population size and w is the growth rate.

Problem: recover the parameters c, w_0, w of the model from an experiment:



A series of n measurements $\{(a_i, t_i)\}_{i=1}^n$.

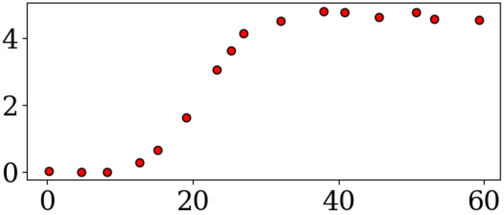
Logistic growth

Limited ressources model: a colony of the bacteria *B. dendroides* is growing in a Petri dish. The colony's area a can be modeled as a function of time t:

$$a(t) = \frac{c}{1 + e^{-wt - w_0}},$$

where c is the carrying capacity, w_0 is the initial population size and w is the growth rate.

Problem: recover the parameters c, w_0, w of the model from an experiment:



A series of n measurements $\{(a_i, t_i)\}_{i=1}^n$.

$$\arg\min_{c, w_0, w} \sum_{i=1}^{n} (a(t_i) - a_i)^2$$

N.B: nonlinear problem!

Solve a nonlinear problem

Transform the model!

$$a(t) = \frac{c}{1 + e^{-wt-w_0}}$$

Solve a nonlinear problem

Transform the model!

$$a(t) = \frac{c}{1 + e^{-wt - w_0}}$$

$$\frac{c}{a(t) - 1} = e^{-wt - w_0}$$

Solve a nonlinear problem

Transform the model!

$$
\begin{aligned}
a(t) &= \frac{c}{1 + e^{-wt-w_0}} \\
\frac{c}{a(t)-1} &= e^{-wt-w_0} \\
\log\left(\frac{c-a(t)}{a(t)}\right) &= -wt - w_0
\end{aligned}
$$

Solve a nonlinear problem

Transform the model!

$$a(t) = \frac{c}{1 + e^{-wt-w_0}}$$

$$\frac{c}{a(t) - 1} = e^{-wt-w_0}$$

$$\log\left(\frac{c - a(t)}{a(t)}\right) = -wt - w_0$$

Ordinary linear regression provided that we have an estimation for c.

Ugly hack: $c := \max_{i \in 1 \dots n} a_i + \varepsilon$

Solve a nonlinear problem

Transform the model!

$$a(t) = \frac{c}{1 + e^{-wt-w_0}}$$
$$\frac{c}{a(t) - 1} = e^{-wt-w_0}$$
$$\log\left(\frac{c - a(t)}{a(t)}\right) = -wt - w_0$$

Ordinary linear regression provided that we have an estimation for c.

Ugly hack: $c := \max\limits_{i \in 1 \dots n} a_i + \varepsilon$

$$\begin{pmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \\ 1 & t_n \end{pmatrix} \begin{pmatrix} w_0 \\ w \end{pmatrix} = \begin{pmatrix} \log \frac{a_1}{c-a_1} \\ \log \frac{a_2}{c-a_2} \\ \vdots \\ \log \frac{a_n}{c-a_n} \end{pmatrix}$$

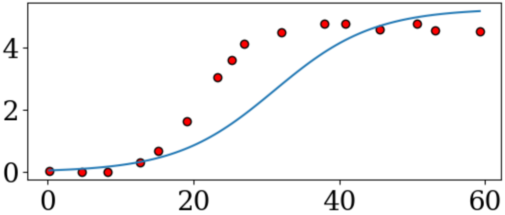
Solve a nonlinear problem

Transform the model!

$$a(t) = \frac{c}{1 + e^{-wt-w_0}}$$

$$\frac{c}{a(t)} - 1 = e^{-wt-w_0}$$

$$\log\left(\frac{c - a(t)}{a(t)}\right) = -wt - w_0$$



Ordinary linear regression provided that we have an estimation for c.

Ugly hack: $c := \max_{i \in 1 \dots n} a_i + \varepsilon$

$$\begin{pmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \\ 1 & t_n \end{pmatrix} \begin{pmatrix} w_0 \\ w \end{pmatrix} = \begin{pmatrix} \log \frac{a_1}{c - a_1} \\ \log \frac{a_2}{c - a_2} \\ \vdots \\ \log \frac{a_n}{c - a_n} \end{pmatrix}$$

Solve a nonlinear problem: linearization

Denote by $\vec{r}(c, w_0, w)$ the residual between the input labels and the predictions:

$$\vec{r}(c, w_0, w) := (a(t_1) - a_1, \dots\ a(t_n) - a_n)$$

Solve a nonlinear problem: linearization

Denote by $\vec{r}(c, w_0, w)$ the residual between the input labels and the predictions:

$$\vec{r}(c, w_0, w) := (a(t_1) - a_1, \dots\ a(t_n) - a_n)$$

Our problem: $\arg\min\limits_{c, w_0, w} \|\vec{r}(c, w_0, w)\|^2$ **Iterative optimization!** (Gauß–Newton)

Solve a nonlinear problem: linearization

Denote by $\vec{r}(c, w_0, w)$ the residual between the input labels and the predictions:

$$\vec{r}(c, w_0, w) := (a(t_1) - a_1, \dots\ a(t_n) - a_n)$$

Our problem: $\arg\min\limits_{c, w_0, w} \|\vec{r}(c, w_0, w)\|^2$ 💡 **Iterative optimization!** (Gauß–Newton)

Denote by $\vec{b} := (c, w_0, w)$ the vector of unknown parameters.

Start from an initial guess $\vec{b}^{(0)}$, build a sequence of approximations $\vec{b}^{(k+1)} = \vec{b}^{(k)} + \vec{\Delta}^{(k)}$

Solve a nonlinear problem: linearization

Denote by $\vec{r}(c, w_0, w)$ the residual between the input labels and the predictions:

$$\vec{r}(c, w_0, w) := (a(t_1) - a_1, \dots \ a(t_n) - a_n)$$

Our problem: $\arg\min\limits_{c, w_0, w} \|\vec{r}(c, w_0, w)\|^2$ 💡 **Iterative optimization!** (Gauß–Newton)

Denote by $\vec{b} := (c, w_0, w)$ the vector of unknown parameters.
Start from an initial guess $\vec{b}^{(0)}$, build a sequence of approximations $\vec{b}^{(k+1)} = \vec{b}^{(k)} + \vec{\Delta}^{(k)}$

Linearize the function $\vec{r}$ at the point $\vec{b}^{(k)}$:

$$\vec{r}(\vec{b}) \approx \vec{r}\left(\vec{b}^{(k)}\right) + J\vec{r}\left(\vec{b}^{(k)}\right) \cdot \left(\vec{b} - \vec{b}^{(k)}\right)$$

Solve a nonlinear problem: linearization

Denote by $\vec{r}(c, w_0, w)$ the residual between the input labels and the predictions:

$$\vec{r}(c, w_0, w) := (a(t_1) - a_1, \dots\ a(t_n) - a_n)$$

Our problem: $\arg\min\limits_{c,w_0,w} \|\vec{r}(c, w_0, w)\|^2$ 💡 **Iterative optimization!** (Gauß–Newton)

Denote by $\vec{b} := (c, w_0, w)$ the vector of unknown parameters.
Start from an initial guess $\vec{b}^{(0)}$, build a sequence of approximations $\vec{b}^{(k+1)} = \vec{b}^{(k)} + \vec{\Delta}^{(k)}$

Linearize the function $\vec{r}$ at the point $\vec{b}^{(k)}$:

$$\vec{r}(\vec{b}) \approx \vec{r}\left(\vec{b}^{(k)}\right) + J\vec{r}\left(\vec{b}^{(k)}\right) \cdot \left(\vec{b} - \vec{b}^{(k)}\right)$$

$$\Delta^{(k)} \leftarrow \arg\min_{\Delta^{(k)}} \left\| J\vec{r}\left(\vec{b}^{(k)}\right) \Delta^{(k)} - \vec{r}\left(\vec{b}^{(k)}\right) \right\|^2$$

Good news: ordinary linear regression

Solve a nonlinear problem: linearization

Denote by $\vec{r}(c, w_0, w)$ the residual between the input labels and the predictions:

$$\vec{r}(c, w_0, w) := (a(t_1) - a_1, \dots\ a(t_n) - a_n)$$

Our problem: $\arg\min\limits_{c,w_0,w} \|\vec{r}(c, w_0, w)\|^2$ 💡 **Iterative optimization!** (Gauß–Newton)

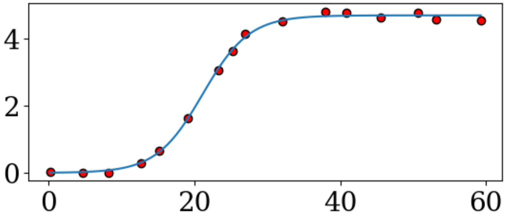
Denote by $\vec{b} := (c, w_0, w)$ the vector of unknown parameters.
Start from an initial guess $\vec{b}^{(0)}$, build a sequence of approximations $\vec{b}^{(k+1)} = \vec{b}^{(k)} + \vec{\Delta}^{(k)}$

Linearize the function $\vec{r}$ at the point $\vec{b}^{(k)}$:

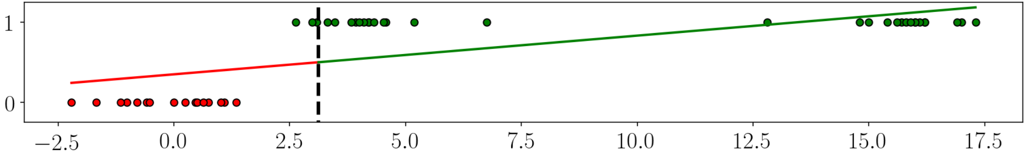
$$\vec{r}(\vec{b}) \approx \vec{r}\left(\vec{b}^{(k)}\right) + J\vec{r}\left(\vec{b}^{(k)}\right) \cdot \left(\vec{b} - \vec{b}^{(k)}\right)$$

$$\Delta^{(k)} \leftarrow \arg\min_{\Delta^{(k)}} \left\| J\vec{r}\left(\vec{b}^{(k)}\right) \Delta^{(k)} - \vec{r}\left(\vec{b}^{(k)}\right) \right\|^2$$

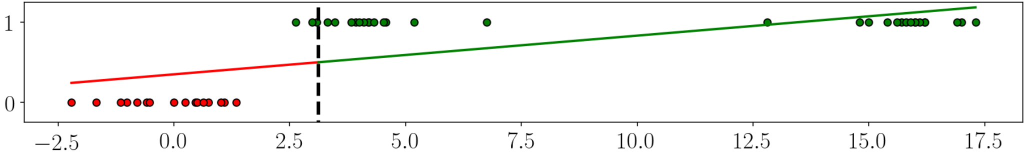
Good news: ordinary linear regression



Back to the classification: quadratic loss function



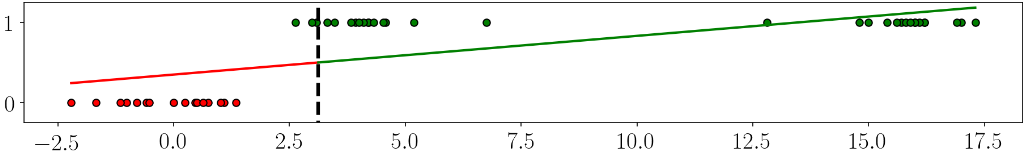
Back to the classification: quadratic loss function



Fit a sigmoid! $1/(1+e^{-wx-w_0})$

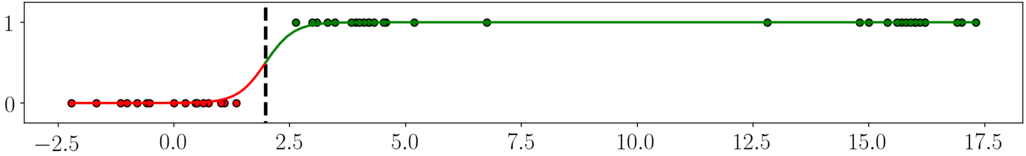
Leave the same decision rule: $y(x) > 1/2$ then x is green, red otherwise.

Back to the classification: quadratic loss function



💡 **Fit a sigmoid!** $1/(1 + e^{-wx - w_0})$
Leave the same decision rule: $y(x) > 1/2$ then x is green, red otherwise.

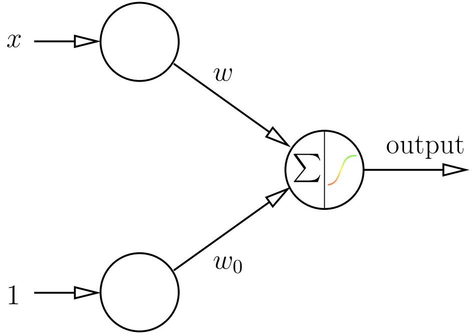
Two parameters only, call Gauß–Newton:



N.B: attention to overfitting! May need to regularize the objective function.

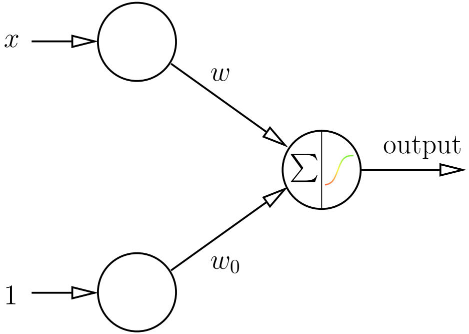
Good news, bad news

Good news:
we have just trained a neural network

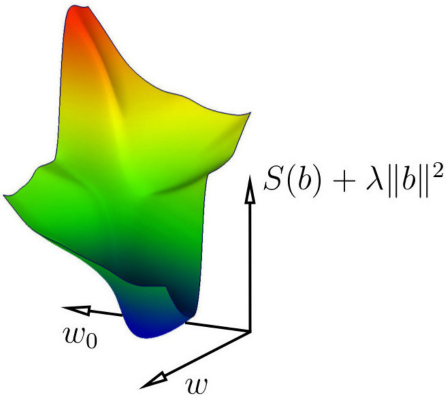


Good news, bad news

Good news:
we have just trained a neural network



Bad news:
the energy is not convex :(



Cross-entropy loss function

As before, let two possible classes encoded as $y \in \{0, 1\}$ and assume

$$p(y = 1|x, w) := \frac{1}{1 + e^{-w^\top x}},$$

where w is a $(m + 1)$-parameter vector and the last element of x is the constant 1.

Cross-entropy loss function

As before, let two possible classes encoded as $y \in \{0, 1\}$ and assume

$$p(y = 1|x, w) := \frac{1}{1 + e^{-w^\top x}},$$

where w is a $(m + 1)$-parameter vector and the last element of x is the constant 1.

Fit the parameters of the sigmoid to match the data. Up to this point nothing has changed!

Cross-entropy loss function

As before, let two possible classes encoded as $y \in \{0, 1\}$ and assume

$$p(y = 1|x, w) := \frac{1}{1 + e^{-w^\top x}},$$

where w is a $(m+1)$-parameter vector and the last element of x is the constant 1.

Fit the parameters of the sigmoid to match the data. Up to this point nothing has changed!

The game changer: Bernoulli's scheme

$\log \mathcal{L}(w) = \log \prod_{i=1}^n p_i(w)^{y_i}(1 - p_i(w))^{1-y_i}$, where $p_i(w) := p(y_i = 1|x_i, w)$.

Problem: $\arg\max_{w} \log \mathcal{L}(w)$

Refer to the course notes for all details of the derivation.

Cross-entropy loss function

Problem: $\arg\max_{w} \log \mathcal{L}(w)$

Cross-entropy loss function

Problem: $\arg\max\limits_{w} \log \mathcal{L}(w) \quad \Leftrightarrow \quad \frac{\partial \log \mathcal{L}}{\partial w}(w) = X^\top (y - p) = 0$, where

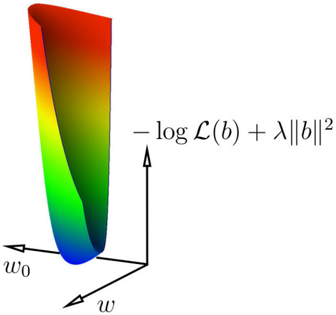
$X := \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^\top$, $y := \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\top$ and $p := \begin{bmatrix} p_1(w) & \dots & p_n(w) \end{bmatrix}^\top$.

Cross-entropy loss function

Problem: $\arg\max_w \log \mathcal{L}(w) \quad \Leftrightarrow \quad \frac{\partial \log \mathcal{L}}{\partial w}(w) = X^\top (y - p) = 0$, where

$X := \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^\top$, $y := \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\top$ and $p := \begin{bmatrix} p_1(w) & \dots & p_n(w) \end{bmatrix}^\top$.

N.B: Non-linear problem, but the Hessian matrix $\frac{\partial^2 \log \mathcal{L}}{\partial w \partial w^\top}$ is definite positive, so the problem is **convex**!

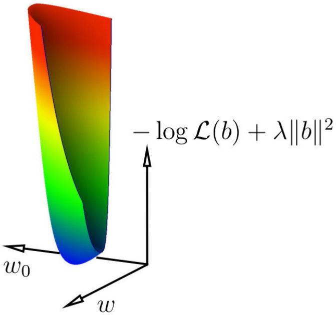


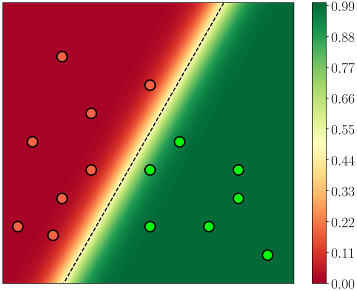
Cross-entropy loss function

Problem: $\arg\max_{w} \log \mathcal{L}(w) \quad \Leftrightarrow \quad \frac{\partial \log \mathcal{L}}{\partial w}(w) = X^\top (y - p) = 0$, where

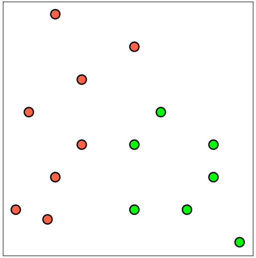
$X := \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^\top$, $y := \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\top$ and $p := \begin{bmatrix} p_1(w) & \dots & p_n(w) \end{bmatrix}^\top$.

N.B: Non-linear problem, but the Hessian matrix $\frac{\partial^2 \log \mathcal{L}}{\partial w \partial w^\top}$ is definite positive, so the problem is **convex**!





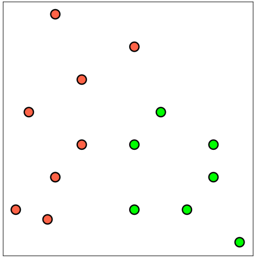
Nonlinear decision boundary: logistic regression



Linear decision boundary:

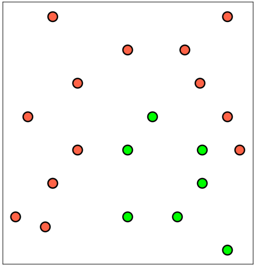
$$\frac{1}{1+e^{-w^\top x}} = \frac{1}{2}$$

Nonlinear decision boundary: logistic regression



Linear decision boundary:

$$\frac{1}{1+e^{-w^\top x}} = \frac{1}{2}$$



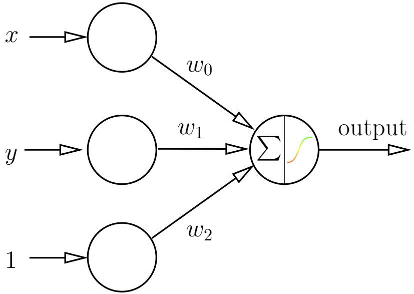
Quadratic decision boundary:

$$\frac{1}{1+e^{-x^\top A x}} = \frac{1}{2}$$

But where is the fun in that...

Nonlinear decision boundary: 3 neurons

Linear decision boundary:

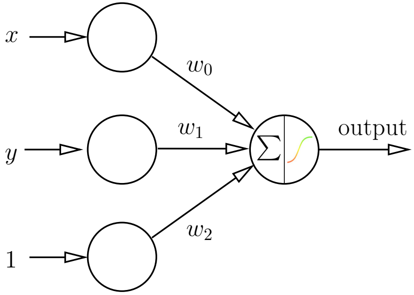


$$\arg\min_{w} \sum_{i=1}^{n} \left(\sigma\left(w^\top x_i\right) - y_i\right)^2,$$

where $\sigma(x, w) := \frac{1}{1+e^{-w^\top x}}$

Nonlinear decision boundary: 3 neurons

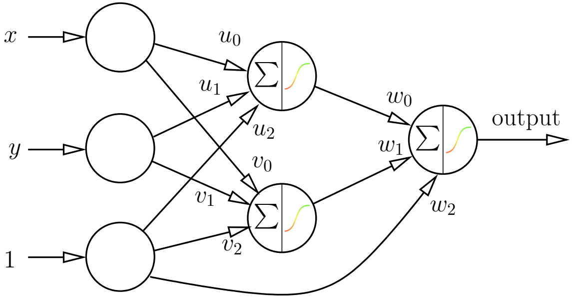
Linear decision boundary:



$$\arg\min_{w} \sum_{i=1}^{n} \left(\sigma\left(w^\top x_i\right) - y_i\right)^2,$$

where $\sigma(x, w) := \frac{1}{1+e^{-w^\top x}}$

Nonlinear decision boundary:

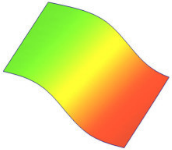


$$\arg\min_{u,v,w} \sum_{i=1}^{n} \left(\sigma\left(w^\top x'_i\right) - y_i\right)^2,$$

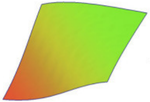
where $x'_i := \begin{pmatrix}\sigma(u^\top x_i) & \sigma(v^\top x_i) & 1\end{pmatrix}^\top$

Nonlinear decision boundary: 3 neurons

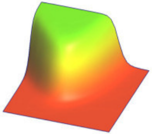
$\circ \left(w_0 \cdot \right.$



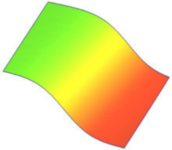
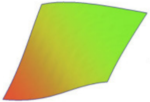
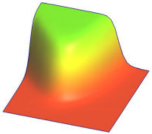
$+ \; w_1 \cdot$

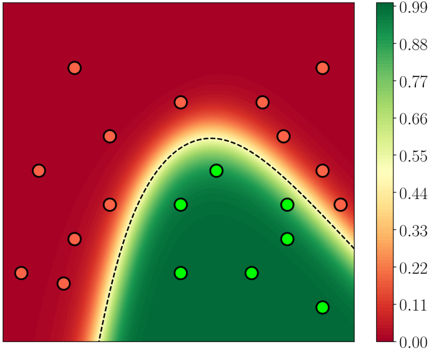


$\left. + \; w_2 \right) =$



Nonlinear decision boundary: 3 neurons

 $\circ \Bigg(w_0 \cdot$  $+\ w_1 \cdot$  $+\ w_2 \Bigg) =$ 



Dmitry Sokolov

Least squares for programmers

— with color plates —