Linear and Non-Linear Optimization

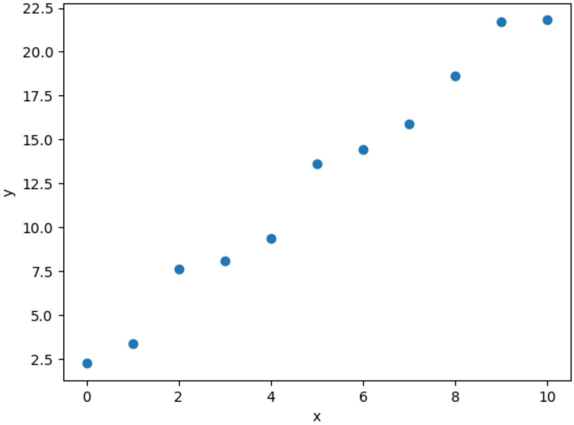
Brent R. Westbrook

March 2, 2022

Linear Regression

Simple Example

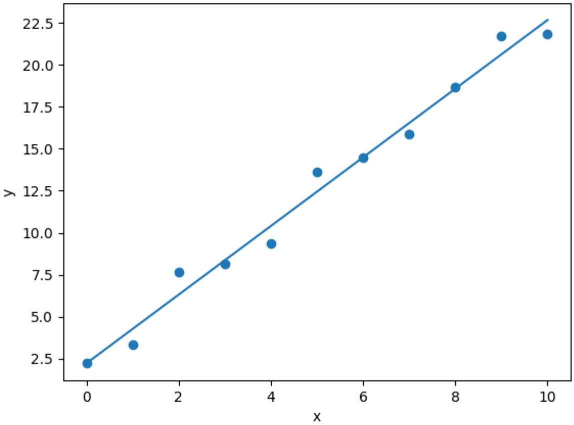
x	y
0.0	2.3
1.0	3.4
2.0	7.6
3.0	8.1
4.0	9.4
5.0	13.6
6.0	14.5
7.0	15.9
8.0	18.6
9.0	21.7
10.0	21.8



Linear Regression

Simple Example

x	y
0.0	2.3
1.0	3.4
2.0	7.6
3.0	8.1
4.0	9.4
5.0	13.6
6.0	14.5
7.0	15.9
8.0	18.6
9.0	21.7
10.0	21.8



Linear Regression

How does it work?

Black box

```
z = np.polyfit(xs, ys, 1)
# => [2.04 2.23]
```

Linear Regression
How does it work?

Black box

```
z = np.polyfit(xs, ys, 1)
# => [2.04 2.23]
```

Math

Solve

$$Ax = b$$

for matrix A and vectors x and b

Two Cases

Exact solution

System of Equations

$$\begin{aligned} x + 4y &= 2 \\ 2x + 5y &= -2 \end{aligned}$$

Two Cases

Exact solution

System of Equations

$$x + 4y = 2$$
$$2x + 5y = -2$$

Matrix Form

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

Two Cases

Exact solution

System of Equations

$$x + 4y = 2$$
$$2x + 5y = -2$$

Matrix Form

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

Code

```
import numpy as np

A = np.array([1, 4, 2, 5]).reshape(2, 2)
b = np.array([2, -2])
soln = np.linalg.solve(A, b)
# => [-6, 2]
```

Two Cases

No exact solution

Idea

Instead of solving $Ax = b$ exactly,
minimize $||Ax - b||$

Two Cases

No exact solution

Idea

Instead of solving $Ax = b$ exactly, minimize $||Ax - b||$

Equation of a line

$y = mx + b$

Matrix version

$$\begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = y$$

Two Cases

No exact solution

Back to the trend line

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \\ 9 & 1 \\ 10 & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} 2.3 \\ 3.4 \\ 7.6 \\ 8.1 \\ 9.4 \\ 13.6 \\ 14.5 \\ 15.9 \\ 18.6 \\ 21.7 \\ 21.8 \end{bmatrix}$$

```
import numpy as np

x = [
 0.0, 1.0, 2.0,
 3.0, 4.0, 5.0,
 6.0, 7.0, 8.0,
 9.0, 10.0,
]
ones = [1.0 for i in x]
A = np.stack((x, ones)).transpose()
y = [
 2.3, 3.4, 7.6,
 8.1, 9.4, 13.6,
 14.5, 15.9, 18.6,
 21.7, 21.8,
]
soln = np.linalg.lstsq(A, y)
```

Solution

[2.04, 2.23], same as before

Why does this matter?

Why does this matter?

Generalization!

Why does this matter?

Generalization!

- Already seen exact vs least-squares solution
  - more rows in A
- Extends to more variables
  - more columns in A
- Extends to polynomials

Why does this matter?

Generalization!

- Already seen exact vs least-squares solution
  - more rows in A
- Extends to more variables
  - more columns in A
- Extends to polynomials

Polynomial regression

$$A = \begin{bmatrix} x_1^m & \dots & x_1^2 & x_1 & 1 \\ x_2^m & \dots & x_2^2 & x_2 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_n^m & \dots & x_n^2 & x_n & 1 \end{bmatrix}$$

Each column has a different power of x in addition to coefficient

ANPASS is just polynomial regression

QFF Equation

$$V = \frac{1}{2}\sum_{ij} F_{ij}\Delta_i\Delta_j + \frac{1}{6}\sum_{ijk} F_{ijk}\Delta_i\Delta_j\Delta_k + \frac{1}{24}\sum_{ijkl} F_{ijkl}\Delta_i\Delta_j\Delta_k\Delta_l$$

ANPASS is just polynomial regression

QFF Equation

$$V = \frac{1}{2}\sum_{ij} F_{ij}\Delta_i\Delta_j + \frac{1}{6}\sum_{ijk} F_{ijk}\Delta_i\Delta_j\Delta_k + \frac{1}{24}\sum_{ijkl} F_{ijkl}\Delta_i\Delta_j\Delta_k\Delta_l$$

Matrix Version

$$XF = V$$

- V is a vector of energies
- F is a vector of force constants
- X is ... a little more complicated

Matrix form for ANPASS problem

$$X_{ik} = \prod_j x_{ij}^{e_{jk}}$$

where x_{ij} is the jth (horizontal) component of the ith (vertical) displacement

Sample displacments

−0.00500000	−0.00500000	−0.01000000	0.000128387078
−0.00500000	−0.00500000	0.00000000	0.000027809414
−0.00500000	−0.00500000	0.01000000	0.000128387078
−0.00500000	−0.01000000	0.00000000	0.000035977201

and e_{jk} is the jth (row) and kth (column) exponent found at the bottom of the ANPASS input file

Sample exponents

0	1	0	2	1	0	0	3	2	1	0	1	0
0	0	1	0	1	2	0	0	1	2	3	0	1
0	0	0	0	0	0	2	0	0	0	0	2	2

Solving the Problem

Basic version

Just solve like we saw before:

$$XF = V$$

Solving the Problem

Basic version

Just solve like we saw before:

$$XF = V$$

Actually solve

$$F = (X^\intercal X)^{-1} X^\intercal V$$

This gives a safer solution than inverting X directly, but the idea is the same

---

Wikipedia, Polynomial regression

Why this is important

- Matrix formulation let me rewrite ANPASS with more than 20x speedup
- Very useful piece of math

What if the relationships aren't linear?

Non-Linear Least Squares

Goal

"[T]o fit a set of observations with a model that is non-linear in the unknown parameters"

Problem Statement

- Have some function, $f(x, \beta)$, where x is some input and β is a set of parameters.
- Also have a set of "true" values y
- Minimize their difference $y - f(\beta)$

Wikipedia, Non-linear least squares

Non-Linear Least Squares

Gauss-Newton Method

$$(\mathsf{J}^\top \mathsf{J})\delta = \mathsf{J}^\top[\mathsf{y} - \mathsf{f}(\beta)]$$

where

$$\mathsf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial \beta_1} & \cdots & \frac{\partial f_1}{\partial \beta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \beta_1} & \cdots & \frac{\partial f_m}{\partial \beta_n} \end{bmatrix}$$

and δ is the next step in the parameters β

Non-Linear Least Squares

Gauss-Newton Method

$$(\mathsf{J}^\top \mathsf{J})\delta = \mathsf{J}^\top[\mathsf{y} - \mathsf{f}(\beta)]$$

where

$$\mathsf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial \beta_1} & \cdots & \frac{\partial f_1}{\partial \beta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \beta_1} & \cdots & \frac{\partial f_m}{\partial \beta_n} \end{bmatrix}$$

and δ is the next step in the parameters β

- Works okay, but can fail to converge

Non-Linear Least Squares
Gradient Methods

$$\delta = -\left(\frac{\partial \Phi}{\partial \beta_1}, \frac{\partial \Phi}{\partial \beta_2}, \ldots, \frac{\partial \Phi}{\partial \beta_n}, \right)^{\mathsf{T}}$$

Just step in the direction of the gradient

Non-Linear Least Squares

Gradient Methods

$$\delta = -\left(\frac{\partial \Phi}{\partial \beta_1}, \frac{\partial \Phi}{\partial \beta_2}, \ldots, \frac{\partial \Phi}{\partial \beta_n},\right)^{\mathsf{T}}$$

Just step in the direction of the gradient

- Typically converges, but very slowly

General Appearance

$$(\mathsf{J}^\top \mathsf{J} + \lambda \mathsf{I})\delta = \mathsf{J}^\top [\mathsf{y} - \mathsf{f}(\beta)]$$

Introduces the parameter λ that controls the interpolation between Gauss-Newton and gradient descent

Non-Linear Least Squares

Levenberg-Marquardt

General Appearance

$$(\mathsf{J}^\top \mathsf{J} + \lambda \mathsf{I})\delta = \mathsf{J}^\top [\mathsf{y} - \mathsf{f}(\beta)]$$

Introduces the parameter λ that controls the interpolation between Gauss-Newton and gradient descent

Basic steps

- Compute J with finite differences
- Solve for δ

Problem: Gauss-Newton when going well, gradient otherwise

- Introduce the parameter $\nu > 1$
- Let Φ be the norm or measure to minimize and $\Phi^{(r)}$ be the current value
- Compute $\Phi(\lambda)$ and $\Phi(\lambda/\nu)$
  1. If $\Phi(\lambda/\nu) \leq \Phi^{(r)}, let \lambda = \lambda/\nu$
  2. If $\Phi(\lambda/\nu) > \Phi^{(r)}$, and $\Phi(\lambda) \leq \Phi^{(r)}$, let $\lambda = \lambda$
  3. If $\Phi(\lambda/\nu) > \Phi^{(r)}$, and $\Phi(\lambda) > \Phi^{(r)}$, increase λ by ν until for some smallest w, $\Phi(\lambda\nu^{w}) \leq \Phi^{(r)}$

Levenberg-Marquardt

Refinements

Problem: Gauss-Newton when going well, gradient otherwise

- Introduce the parameter $\nu > 1$
- Let Φ be the norm or measure to minimize and $\Phi^{(r)}$ be the current value
- Compute $\Phi(\lambda)$ and $\Phi(\lambda/\nu)$
  1. If $\Phi(\lambda/\nu) \leq \Phi^{(r)}, let \lambda = \lambda/\nu$
  2. If $\Phi(\lambda/\nu) > \Phi^{(r)}$, and $\Phi(\lambda) \leq \Phi^{(r)}$, let $\lambda = \lambda$
  3. If $\Phi(\lambda/\nu) > \Phi^{(r)}$, and $\Phi(\lambda) > \Phi^{(r)}$, increase λ by ν until for some smallest w, $\Phi(\lambda\nu^{w}) \leq \Phi^{(r)}$

What if λ gets unreasonably large?

Levenberg-Marquardt
Refinements

Modify case (3)

Instead of taking step δ, take step $K\delta$, where K is made smaller until $\Phi \leq \Phi^{(r)}$

Modify case (3)

Instead of taking step δ, take step $K\delta$, where K is made smaller until $\Phi \leq \Phi^{(r)}$

When should you do this? (part I had left out)

Modify case (3)

Instead of taking step δ, take step $K\delta$, where K is made smaller until $\Phi \leq \Phi^{(r)}$

When should you do this? (part I had left out)

Angle, γ, between the step and gradient

$$\gamma = \text{acos} \frac{\delta^\intercal g}{(||\delta||)(||g||)}$$

When $\gamma < \frac{\pi}{4}$

Levenberg-Marquardt

Refinements

Problem: Gradient methods are not scale invariant

Transform $J^\intercal J$ (A) into A^*

$$A^* = (a^*_{ij}) = \left(\frac{a_{ij}}{\sqrt{a_{ii}}\sqrt{a_{jj}}}\right)$$

and $J^\top[y - f(\beta)] = g$ into g^*:

$$g^* = (g^*_j) = \left(\frac{g_j}{\sqrt{a_{jj}}}\right)$$

Problem: Gradient methods are not scale invariant

Transform $J^{\intercal}J$ (A) into A^*

$$A^* = (a^*_{ij}) = \left(\frac{a_{ij}}{\sqrt{a_{ii}}\sqrt{a_{jj}}} \right)$$

and $J^{\top}[y - f(\beta)] = g$ into g^*:

$$g^* = (g^*_j) = \left(\frac{g_j}{\sqrt{a_{jj}}} \right)$$

I ran into this when moving from Gaussian to MOPAC, MOPAC parameters vary widely in magnitude

Levenberg-Marquardt

New Issue

Trapped in local minimum?

- γ should be a monotonically decreasing function of λ
- Seems to violate this when stuck or converged ($\gamma \approx 90^\circ$), so just break the loop