# Lecture 4:
# OLS Computation
# Intro To Scraping

## Big Data and Machine Learning for Applied Economics
## Econ 4676

Ignacio Sarmiento-Barbieri

Universidad de los Andes

August 19, 2021

# Agenda

# Class' Repos

- Syllabus Repo
- Lectures Repo
- Problem Set Repo
- Problem Set Template Repo
- e-TAs

# Recap

- Least Square Estimator
- Quick Review of Statistical Properties
- Numerical Properties
- FWL
  - Fixed Effects
  - Leverage
  - Goodness of Fit
  - Updating

# Recap

- The goal here is to solve something which looks like

$$f^{\star} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \{E[L(Y, f(\mathbf{X}))]\} \tag{1}$$

- for some loss function $L$, and for some set of predictors $\mathcal{F}$.
- This is an optimization problem.

# QR decomposition

- Linear Model: Min Risk $\iff$ Min SSR

$$\hat{\beta} = (X'X)^{-1}X'y \tag{2}$$

- Involves inverting a $k \times k$ matrix $X'X$
- requires allocating $O(nk + k^2)$ if n is "big" we cannot store this in memory

Solving directly

```
beta<-solve(t(X)%*%X)%*%t(X)%*%y
```

may not be the smartest move

# QR decomposition

Most software use a QR decomposition

**Theorem** If $A \in \mathbb{R}^{n \times k}$ then there exists an orthogonal $Q \in \mathbb{R}^{n \times n}$ and an upper triangular $R \in \mathbb{R}^{n \times k}$ so that $A = QR$

- Orthogonal Matrices:
  - Def: $Q'Q = QQ' = I$ and $Q' = Q^{-1}$
  - Prop: product of orthogonal is orthogonal, e.g $A'A = I$ and $B'B = I$ then $(AB)'(AB) = B'(A'A)B = B'B = I$
- **(Thin QR)** If $A \in \mathbb{R}^{n \times k}$ has full column rank then $A = Q_1 R_1$ the QR factorization is unique, where $Q_1 \in \mathbb{R}^{n \times k}$ and $R$ is upper triangular with positive diagonal entries

Can use it these to get $\hat{\beta}$

$$(X'X)\hat{\beta} = X'y \tag{3}$$
$$(R'Q'QR)\hat{\beta} = R'Q'y \tag{4}$$
$$(R'R)\hat{\beta} = R'Q'y \tag{5}$$
$$R\hat{\beta} = Q'y \tag{6}$$

Solve by back substitution

# QR decomposition

$$X = \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \quad y = \begin{pmatrix} 1 \\ 4 \\ 2 \end{pmatrix} \tag{7}$$

1. QR factorization X=QR

$$Q = \begin{bmatrix} -0.57 & -0.41 \\ -0.57 & -0.41 \\ -0.57 & 0.82 \end{bmatrix} \quad R = \begin{bmatrix} -1.73 & -4.04 \\ 0 & 0.81 \end{bmatrix} \tag{8}$$

2. Calculate $Q'y = [-4.04, -0.41]'$
3. Solve

$$\begin{bmatrix} -1.73 & -4.04 \\ 0 & 0.81 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} -4.04 \\ -0.41 \end{bmatrix} \tag{9}$$

Solution is $(3.5, -0.5)$

# QR decomposition

This is actually what R does under the hood

| | | |
|---|---|---|
| obj | list [12] (S3: lm) | List of length 12 |
| coefficients | double [2] | -1.71e+08 3.01e+08 |
| residuals | double [207607] | 1.17e+09 -2.38e+08 -5.21e+08 -1.96e+08 -5.12e+07 -1.91e+08 ... |
| effects | double [207607] | -3.15e+11 2.10e+11 -5.24e+08 -1.98e+08 -5.34e+07 -1.93e+08 ... |
| rank | integer [1] | 2 |
| fitted.values | double [207607] | 4.31e+08 4.31e+08 7.32e+08 4.31e+08 4.31e+08 4.31e+08 ... |
| assign | integer [2] | 0 1 |
| qr | list [5] (S3: qr) | List of length 5 |
| df.residual | integer [1] | 207605 |
| xlevels | list [0] | List of length 0 |
| call | language | lm(formula = price ~ bathrooms, data = dta0) |
| terms | formula | price ~ bathrooms |
| model | list [207607 x 2] (S3: data.fra | A data.frame with 207607 rows and 2 columns |

Note that R's `lm` also returns many objects that have the same size as X and Y

# Gradient-Based Optimization

- Suppose we have a function $y = f(x)$, where both x and y are real numbers.
- The derivative $f'(x)$ gives the slope of $f(x)$ at the point x
- It specifies how to scale a small change in the input to obtain the corresponding change in the output

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x) \tag{10}$$

- The derivative is therefore useful for minimizing a function because it tells us how to change x in order to make a small improvement in y
- We can thus reduce $f(x)$ by moving x in small steps with the opposite sign of the derivative.
- This technique is called gradient descent (Cauchy, 1847).

# Gradient-Based Optimization

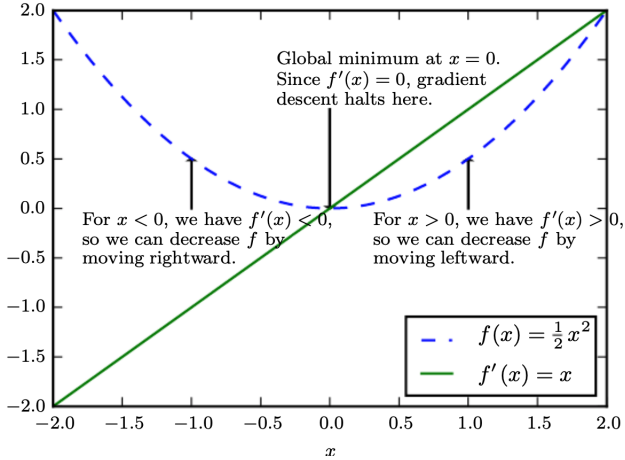



Figure 4.1: Gradient descent. An illustration of how the gradient descent algorithm uses the derivatives of a function to follow the function downhill to a minimum.

# Gradient-Based Optimization

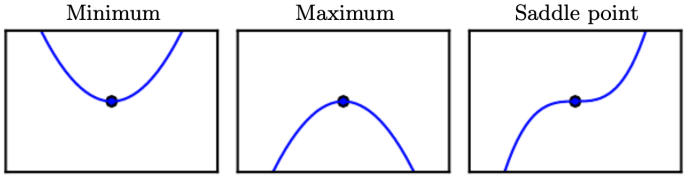



Figure 4.2: Types of critical points. Examples of the three types of critical points in one dimension. A critical point is a point with zero slope. Such a point can either be a local minimum, which is lower than the neighboring points; a local maximum, which is higher than the neighboring points; or a saddle point, which has neighbors that are both higher and lower than the point itself.

# Gradient-Based Optimization

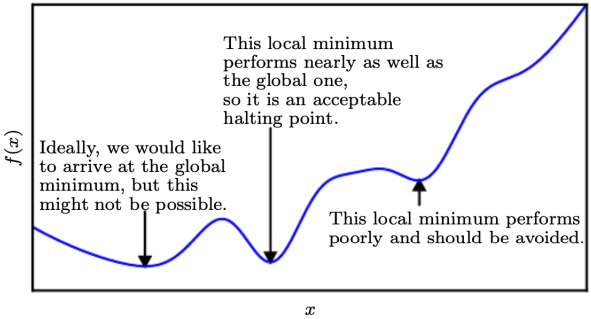


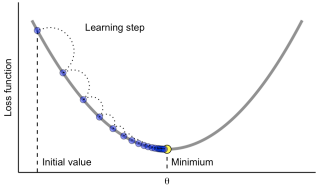

Figure 4.3: Approximate minimization. Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present. In the context of deep learning, we generally accept such solutions even though they are not truly minimal, so long as they correspond to significantly low values of the cost function.

# Gradient-Based Optimization

- **Steepest descent**, or **gradient descent** proposes a new point

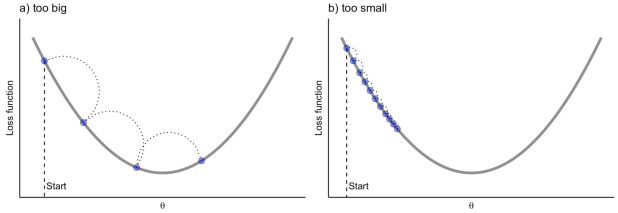
$$\theta' = \theta - \epsilon \nabla_\theta f(\theta) \tag{11}$$

- where $\epsilon$ is the learning rate, a positive scalar determining the size of the step.




Source: Boehmke, B., & Greenwell, B. (2019)

# Gradient-Based Optimization

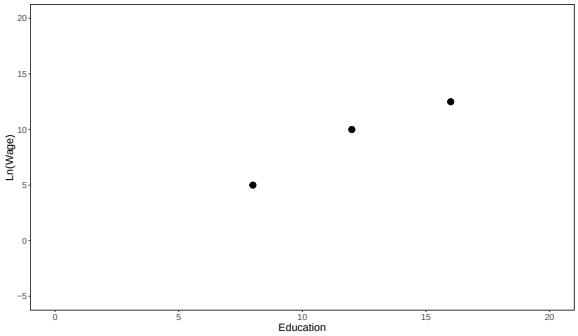



Source: Boehmke, B., & Greenwell, B. (2019)

- We can choose $\epsilon$ in several different ways:
  - A popular approach is to set $\epsilon$ to a small constant.
  - Sometimes, we can solve for the step size that makes the directional derivative vanish.
  - Another approach is to evaluate $f(\theta - \epsilon \nabla_\theta f(\theta))$ for several values of $\epsilon$ and choose the one that results in the smallest objective function value. This is called a line search.
- Steepest descent converges when every element of the gradient is zero (or, in practice, very close to zero).
- In some cases, we may be able to avoid running this iterative algorithm and just jump directly to the critical point by solving the equation $\nabla_\theta f(\theta) = 0$ for x.

# Gradient-Based Optimization

| log(wage) | Education (years) |
|---|---|
| 5 | 8 |
| 10 | 12 |
| 12.5 | 16 |



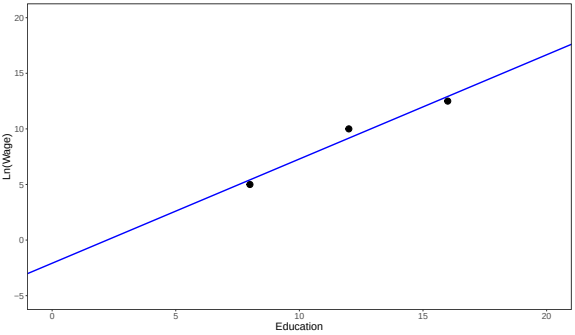
# Gradient-Based Optimization

| log(wage) | Education (years) |
|---|---|
| 5 | 8 |
| 10 | 12 |
| 12.5 | 16 |

$$\hat{\beta} = (X'X)^{-1}X'y$$

```
beta<-solve(t(X)%*%X)%*%t(X)%*%y

lm(y~x,data)
```




$$y = -2.0833 + 0.9375 \times Educ$$

# Gradient-Based Optimization

The Loss Function

$$SSR = f(\theta) = \sum_{i=1}^{n} (y_i - (\alpha + \beta x_i))^2$$

The Gradient

$$\nabla f_\theta(\theta) = \begin{pmatrix} \frac{\partial f}{\partial \alpha} \\ \frac{\partial f}{\partial \beta} \end{pmatrix} = \begin{pmatrix} -2\sum_{i=1}^{n}(y_i - \alpha - \beta x_i) \\ -2\sum_{i=1}^{n} x_i(y_i - \alpha - \beta x_i) \end{pmatrix}$$

Updating

$$\alpha' = \alpha - \epsilon \frac{\partial f}{\partial \alpha}$$
$$\beta' = \beta - \epsilon \frac{\partial f}{\partial \beta}$$

# Gradient-Based Optimization

First Iteration

| log(wage) | Education (years) |
|---|---|
| 5 | 8 |
| 10 | 12 |
| 12.5 | 16 |

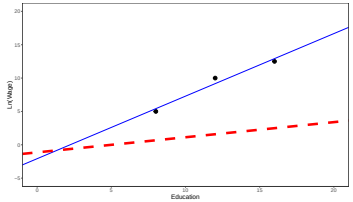
Start with an initial guess: $\alpha = -1; \beta = 2$ , and a learning rate ($\epsilon = 0.005$). Then we have

$$\alpha' = (-1) - 0.005\left(-2\left((5 - (-1) - 2 \times 8) + (10 - (-1) - 2 \times 12) + (12.5 - (-1) - 2 \times 16)\right)\right.$$

$$\beta' = 2 + 0.005\left(-2\left(8(5 - (-1) - 2 \times 8) + 12(10 - (-1) - 2 \times 12) + 16(12.5 - (-1) - 2 \times 16)\right)\right.$$

$$\alpha' = -1.1384$$

$$\beta' = 0.2266$$



# Gradient-Based Optimization

Second Iteration

| log(wage) | Education (years) |
|---|---|
| 5 | 8 |
| 10 | 12 |
| 12.5 | 16 |

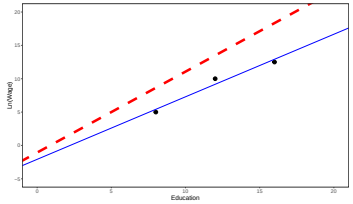
Start with an initial guess: $\alpha = -1; \beta = 2$ , and a learning rate ($\epsilon = 0.005$). Then we have

$$\alpha^2 = (-1.1384) - 0.005\,(-2\,((5 - (-1.1384) - (0.2266) \times 8) + (10 - (-1.1384) - (0.2266) \times 12) + (12.5 - (-1.1384) - (0.2266) \times 16))$$

$$\beta^2 = (0.2266) + 0.005\,(-2\,(8(5 - (-1.1384) - (0.2266) \times 8) + 12(10 - (-1.1384) - (0.2266) \times 12) + 16(12.5 - (-1.1384) - (0.2266) \times 16))$$

$$\alpha^2 = -1.0624$$

$$\beta^2 = 1.212689$$



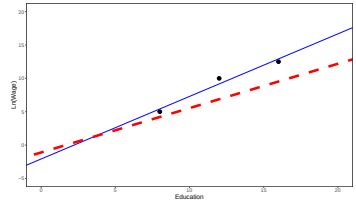
# Gradient-Based Optimization

Third Iteration

| log(wage) | Education (years) |
|---|---|
| 5 | 8 |
| 10 | 12 |
| 12.5 | 16 |

$$\alpha^3 = -1.0624$$

$$\beta^3 = 1.212689$$



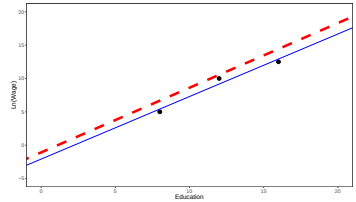
# Gradient-Based Optimization

Fourth Iteration

| log(wage) | Education (years) |
|---|---|
| 5 | 8 |
| 10 | 12 |
| 12.5 | 16 |

$$\alpha^4 = -1.082738$$

$$\beta^4 = 0.9693922$$



# Gradient-Based Optimization

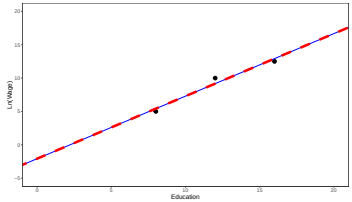
7211 Iteration

| log(wage) | Education (years) |
|---|---|
| 5 | 8 |
| 10 | 12 |
| 12.5 | 16 |

$$\alpha^{7211} = -2.076246$$

$$\beta^{7211} = 0.9369499$$

$$y^{ols} = -2.0833 + 0.9375 \times Educ$$



# Stochastic Gradient-Based Optimization

- Computing the gradient can be very time consuming.
- However, often it is possible to find a "cheap" approximation of the gradient.

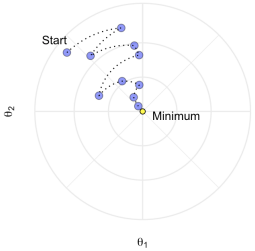
$$SSR = f(\theta) = \sum_{i=1}^{n} (y_i - (\alpha + \beta x_i))^2$$

$$\theta_{j+1} = \theta_j - \epsilon_j \sum_{i=1}^{n} (\nabla_\theta f_i(\theta_j))$$

- Approximating the gradient is still useful as long as it points in roughly the same direction as the true gradient.
- We randomly typically without replacement) choose a subset of $\nabla_\theta f_i(\theta)$ for mini-batch gradient descent (mini-batch can be one)

# Stochastic Gradient-Based Optimization

- The key insight we only need taht $E(\nabla_\theta f_i(\theta_j)) = \nabla_\theta f(\theta)$
- The word stochastic here refers to the fact that we acknowledge that we do not know the gradient precisely, but instead only know a noisy approximation to it.

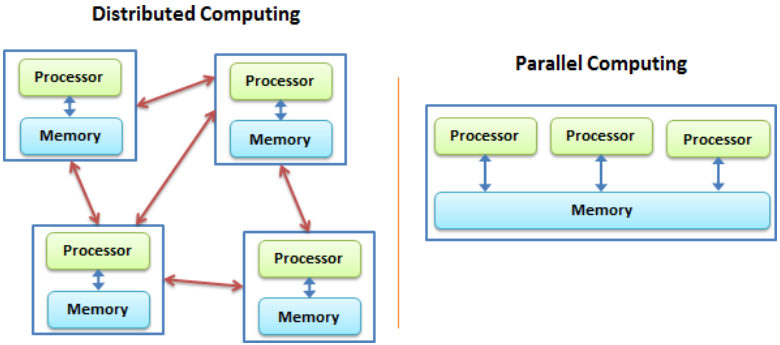



Source: Boehmke, B., & Greenwell, B. (2019)

- This makes the algorithm faster but
- Adds some random nature in descending the loss function's gradient.
- Although this randomness does not allow the algorithm to find the absolute global minimum, it can actually help the algorithm jump out of local minima and off plateaus to get sufficiently near the global minimum.

# Parallel vs Distributed

- An algorithm is parallel if it does many computations at once.
  - It needs to see all of the data
- It is distributed if you can work with subsets of data
  - `Stata-mp` is parallel. (license charges by core)
  - `R` and `Python` can be parallel **and** distributed

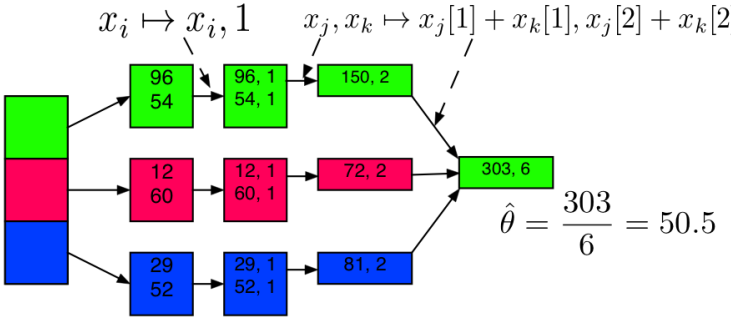



https://tinyurl.com/y3nzvkwh

# Map Reduce

- Original Paper *MapReduce: Simplified data processing on large clusters (2004) Dean and Ghemawat*
- It is of the most popular frameworks
- Basic Idea:
  1. You need to be able to specify a key that indexes subgroups of data that can be analyzed in isolation.
  2. Map: Calculate and sort relevant statistics by key
  3. Partition and pipe the outcome of map so that outcomes with the same key end up on the same machine
  4. Reduce: Apply a summarization operation within the subgroup defined by each key.

# Example: Mean by groups




https://datascienceguide.github.io/map-reduce

# QR decomposition for block matrices

Idea on how to distribute OLS (Constantine & Gleich, 2011)

$$X_{8n\times k} = \begin{bmatrix} X^1_{2n\times k} \\ X^2_{2n\times k} \\ X^3_{2n\times k} \\ X^4_{2n\times k} \end{bmatrix} \tag{12}$$

QR to each block

$$X_{8n\times k} = \underbrace{\begin{bmatrix} Q^1_{2n\times k} & & & \\ & Q^2_{2n\times k} & & \\ & & Q^3_{2n\times k} & \\ & & & Q^4_{2n\times k} \end{bmatrix}}_{8n\times 4k} \underbrace{\begin{bmatrix} R^1_{k\times k} \\ R^2_{k\times k} \\ R^3_{k\times k} \\ R^4_{k\times k} \end{bmatrix}}_{4k\times k} \tag{13}$$

$$X_{8n\times k} = \underbrace{\underbrace{\begin{bmatrix} Q^1_{2n\times k} & & & \\ & Q^2_{2n\times k} & & \\ & & Q^3_{2n\times k} & \\ & & & Q^4_{2n\times k} \end{bmatrix}}_{8n\times 4k} \underbrace{Q_2}_{4k\times k} \underbrace{R_2}_{k\times k}}_{Q} \tag{14}$$

# Spark

- The tools facilitating distributed computing are rapidly improving.
- One prominent system is Spark, that is quickly replacing MapReduce
- Seamlessly integration with `R` and `Python` and has it's own `MLlib`
  - E.g. Spark uses distributed version of stochastic gradient descent to compute OLS
- One of the key differences with MapReduce is how they load data
  - MapReduce has to read from and write to a disk
  - Spark loads it in-memory (can get 100x faster)

# Review & Next Steps

- Computation
- QR decomposition
- Gradient Descent
- MapReduce and Spark

- **Next Class:** Problem Set Presentations

- Questions? Questions about software?

# Further Readings


- Boehmke, B., & Greenwell, B. (2019). Hands-on machine learning with R. Chapman and Hall/CRC.
- Constantine, P. G., & Gleich, D. F. (2011, June). Tall and skinny QR factorizations in MapReduce architectures. In Proceedings of the second international workshop on MapReduce and its applications (pp. 43-50).
- Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters.
- Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). Mathematics for machine learning. Cambridge University Press.
- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT press.
- Van Loan, C. F., Golub, G. H. (2012). Matrix Computations. United States: Johns Hopkins University Press.
- `Webscraping` tutorial from Prof. Grant McDermott.
- Web Scrapping slides from Fernandez Villaverde J., Guerrón P. & Zarruk Valencia, D.