The University of Texas at Austin
Department of Electrical and Computer Engineering

EE380L: Data Mining — Spring 2016

PROBLEM SET ONE SOLUTIONS

Constantine Caramanis .

This problem set gives some practice/refresher with Python. It also has us play with some regression examples, and then some abstract 1-D and polynomial regression. Finally, we learn about gradient descent in 1 and higher dimensions, and about eigenvectors and eigenvalues, and their role in rates of convergence of gradient descent.

1. Download this dataset about professors in CS departments:
`http://users.ece.utexas.edu/~cmcaram/EE380L-2016/cs52.csv`

   Use Pandas to load this dataset in Python and count many professors teaching in UC Berkeley obtained their Ph.D. in UC Berkeley. Submit your python code through canvas. Name your file hw1.py and assume that cs52.csv will be in the same folder.

2. This problem gives some more practice with Python, and gets us started with Regression. More importantly, it gets to the heart of a most important question: *Would we be better off just watching TV?*

   Back in the 70s, there was a popular educational TV series called *The Electric Company* (`https://en.wikipedia.org/wiki/The_Electric_Company`). In a number of schools in Fresno and Youngstown, the benefits of watching this TV show as either a *supplement* to the curriculum, or as a *replacement* to part of the curriculum, was explored, using students from 192 classrooms across 4 grades.

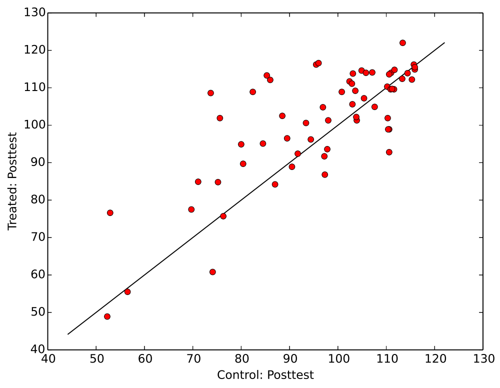
   Download the data set `electric.dat`, which you'll find in the `electric.company` subfolder of the collection of data sets you can find here `http://www.stat.columbia.edu/~gelman/arm/examples/ARM_Data.zip`.[1] For this exercise, we will only use part of these data.

   The data set contains the results from 100 collections of students in two different towns in 4 different grades. These students were split up into a control group, and a treatment group. The treatment group was further split into students whose curriculum was Supplemented (denoted by 'S' in the final column in the data set) by the TV show, and students whose curriculum was Replaced (denoted by 'R' in the final column in the data set) by the TV show. Aggregate reading scores were measured for each group at the end of the year.

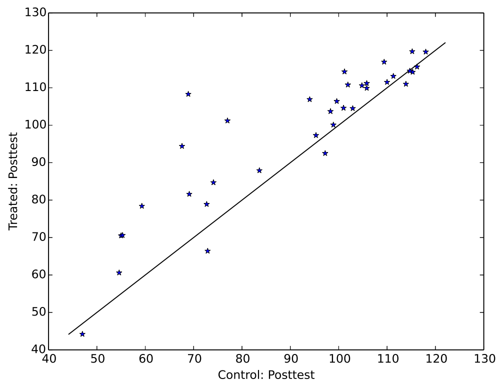
   (a) Plot `treated.Posttest` vs. `control.Posttest` for all students (so across all grades and cities) in the 'Supplement' group, and find the sample average of Treated.Posttest - Control.Posttest for these students.
   Solution Sample average of the difference: 4.825 (See Fig. 1).

   (b) Do the same for all the students in the Replacement group.
   Solution Sample average of the difference: 7.109 (See Fig. 1).

[1]This is the data that accompanies the book *Data Analysis Using Regression and Multilevel/Hierarchical Models*, by Gelman and Hill.



i) Supplement group



ii) Replacement group

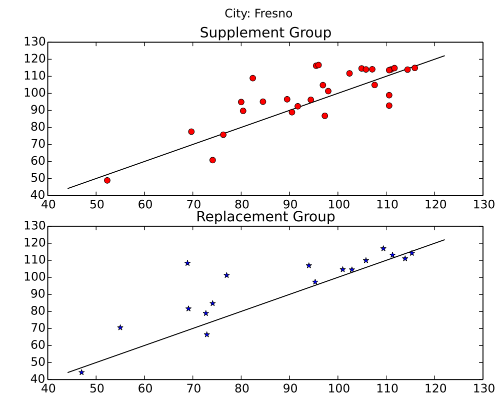
Figure 1: Would it be better off?: `treated.Posttest` vs. `control.Posttest`.

(c) Now to decide which was better: Compare the above plots. Do so by considering a line $y = x$. Are most of the points above or below that line? Using your observation of that, would you favor Supplementing or Replacing?

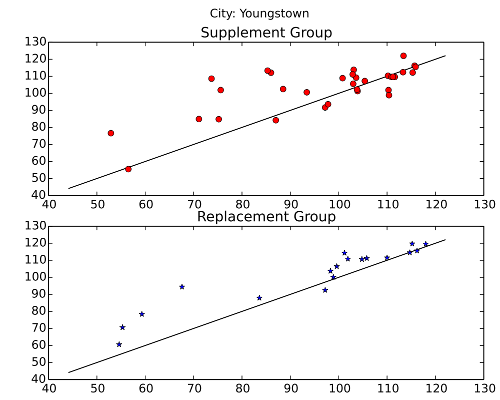
Solution Replacing.

(d) (Bonus) Are the results uniform across the student groups? Is there some group that benefited more or less?

Solution Yes. We can see from Fig. 2 and Fig. 3 that the results are uniform across the student groups.

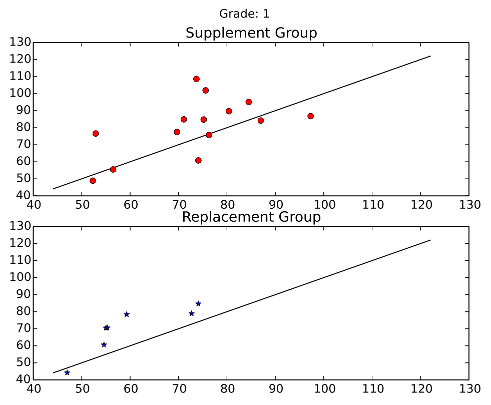


i) Students in Fresno

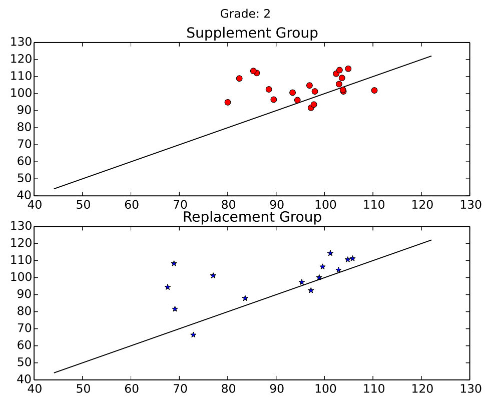


ii) Students in Youngstown

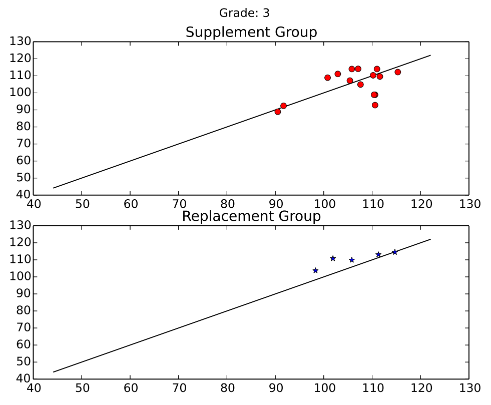
Figure 2: Achievement gap across different student groups based on: City.



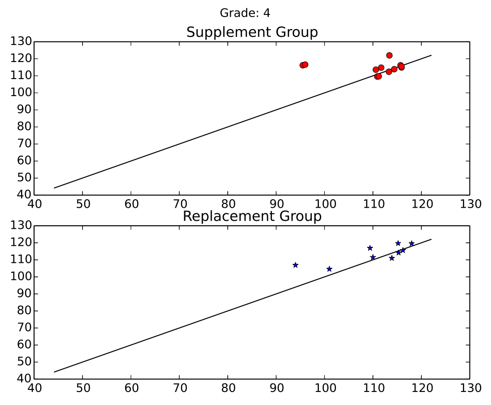
i) First grade students



ii) Second grade students



iii) Third grade students



iv) Fourth grade students

Figure 3: Achievement gap across different student groups based on: Grade.

3. This problem illustrates the fact that blindly applying tools from machine learning is not always a wise idea. We illustrate this through the simple idea of *logarithmic transformations*. Here's a short read on this that illustrates the basic point with a few more examples like the one we explore here `http://www.kenbenoit.net/courses/ME104/logmodels2.pdf`.

   From the same book/authors as in the above problem, find and download the `pollution.dta` file here `http://www.stat.columbia.edu/~gelman/arm/examples/pollution/`. Note that this is a .dta file, not a .dat file. You'll need a different `pandas` command to read this.

   These data capture mortality rates and other environmental factors across 60 U.S. metropolitan areas. In the name of oversimplifying to make a point... let's consider only mortality rate (last column) versus the concentration of nitric oxides (fourth to last column).

   (a) Plot (as a scatterplot) mortality vs the level of nitric oxides. Would you expect linear regression to provide a good fit of these data?

   Solution We cannot expect the linear regression performs well because the relationship between mortality and the level of nitric oxides does not look linear as shown in Fig. 4.

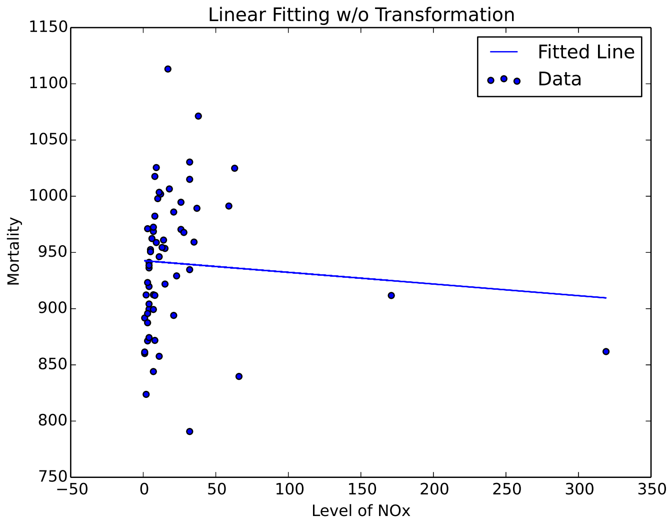


Figure 4: A scatter plot of mortality vs. the level of nitric oxides and the fitted line of the data.

   (b) Check your answer: Fit a line of the form

$$y = \beta_1 x + \beta_0$$

   to the data, using, as we did in class, to minimize the squared error, aka the squared loss. That is: find the least squares solution. Thus, the output variable, y_i, will be mortality of city $i \in \{0, \ldots, 59\}$, and the input variable, x_i, represents the level of NOx in city i. Note that you are searching for 2 variables, β_1 and β_0. You can do this using `scipy.linalg.lstsq`, which we also used in the IPython notebook from class (see `http://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.linalg.lstsq.html`). Unlike the regression packages from `scikit learn` which we'll be playing with later, this more basic package requires you to write the problem as

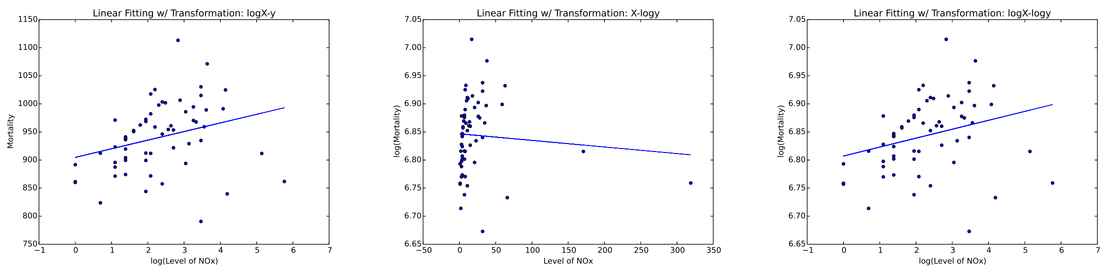
$$\min: \ \|X\beta - y\|_2^2.$$

(Here, β will be the vector (β_1, β_0).) Note that this is precisely what we did in class, first for the scalar case, and then for fitting a second order polynomial.

Solution See Fig. 4.

(c) Now, find a transformation of the data that gives something that is a bit closer to a linear relationship (and hence one where linear regression is a good idea). Then repeat your plotting and regression of the above.

Solution We can see from Fig. 5 that the *linear-log* transformation $(y - \log X)$ leads to a (slightly) better linear relationship. We can also check that the traning error of the *linear-log* transformation is (slightly) smaller than those of other transformations.



i) $y = \beta_1 \log x + \beta_0$ ii) $\log y = \beta_1 x + \beta_0$ iii) $\log y = \beta_1 \log x + \beta_0$

Figure 5: Linear regression with logaritmic transforms.

4. In this exercise, you will write a program that given k points $(x_i, y_i) \in \mathbb{R} \times \mathbb{R}$, $i = 1, \ldots, k$, finds the minimum degree polynomial that goes through each point.

(a) Use `numpy.polyfit` to do this. If your points are generated at random, what do you observe about the degree of the polynomial you need, and the number of points you have?

Solution We need a polynomial of degree at least $k-1$ to fit the relationship of k points.

(b) Show that if you have k points and a polynomial of degree k, finding the exact fit, i.e., the degree-k polynomial that goes through the k points exactly, amounts to solving a system of linear equations: $Ax = b$.

Solution Let the degree-k polynomial be $f(x) = a_0 + a_1 x + \cdots + a_k x^k$. Then we can write the relationship of (x_i, y_i) for $i = 1, \cdots, k$ as

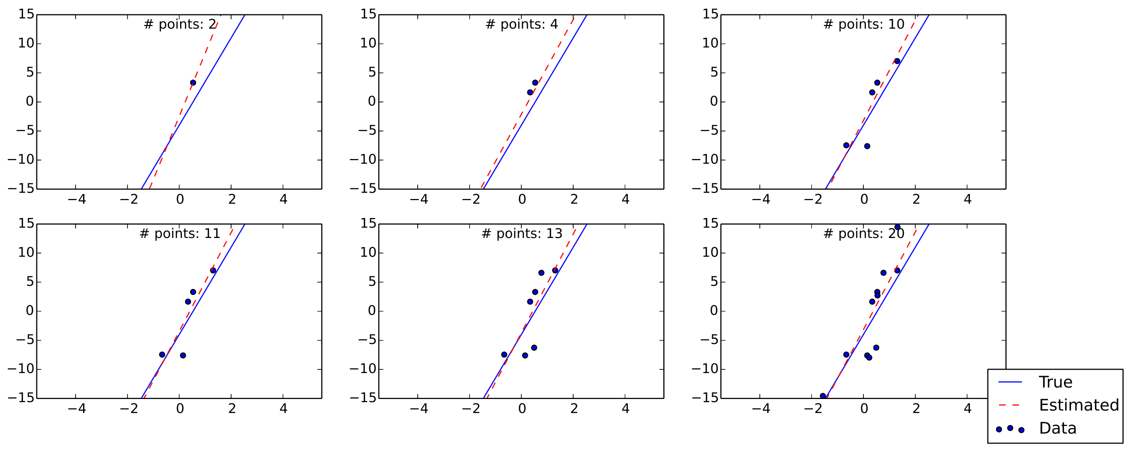
$$\begin{aligned}
y_1 &= a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_k x_1^k \\
y_2 &= a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_k x_2^k \\
&\vdots \\
y_k &= a_0 + a_1 x_k + a_2 x_k^2 + \cdots + a_k x_k^k
\end{aligned}$$

Since we have $k+1$ unknowns $a_0, a_1, \cdots a_k$ and k linear equations, we have infitnitely many solutions if the equations are consistent. Therefore, if we have k points, we can find a degree-k polynomial that goes through all points exactly.

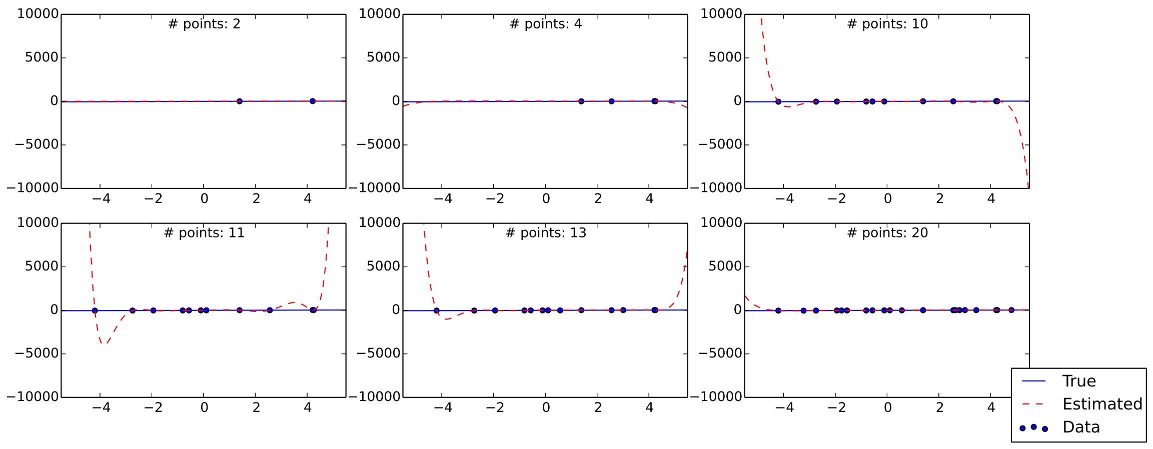
(c) (Bonus) As we did in class, experiment with overfitting vs amount of data. Generate points according to a line, but then add a little noise. Then try fitting a line (using least squares). Generate more and more points according to that same relationship (always adding noise) and notice that the line that you find, very quickly stabilizes and does not change. Now try fitting a much higher degree, say, 10 or 15, polynomial. Note that it takes many more points before the produced solution stops changing. In particular, note how much the solution changes between the 10 and 11 (or so...) points that you generate and fit.

The important moral of this experiment/game, is one of the hallmarks of overfitting. If your algorithm is *unstable*, in the sense that with slightly perturbed data, or a few more/less data points, it produces a very different solution, then you should be highly suspicious that you are overfitting. If, on the other hand, your algorithm is *stable*, i.e., the solution it produces depends on all or many of the points, and hence does not change much when you change just one or two points, then it's quite likely that you are not overfitting.

Solution See Fig. 6.



i) Line fitting



ii) Degree-10 polynomial fitting

Figure 6: Stabilization of polynomial fitting as the number of used data changes.

5. Recall that we spent some time discussing *Euclidean Distance*, which we also referred to as the ℓ^2-norm. For a vector $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$, we denoted and defined this as follows:

$$\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}}.$$

More generally we can define the ℓ^p norm for $1 < p < \infty$ as follows: For a vector $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$,

$$\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}.$$

Additionally, for $p = \infty$, the so-called *infinity norm* is defined as

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i|.$$

(a) Plot the *unit ball* with respect to the p-norm for $p = 1, 2, 10, \infty$, in two dimensions, $n = 2$:

$$B_{\ell^p} = \{\boldsymbol{x} = (x_1, x_2) \, : \, \|\boldsymbol{x}\|_p \leq 1\}.$$

One way to do this is by generating random points in the square $[-1, 1]^n$ and plotting those that have norm less than 1. You can use the `numpy` command `linalg.norm.`
Solution See Fig. 7.

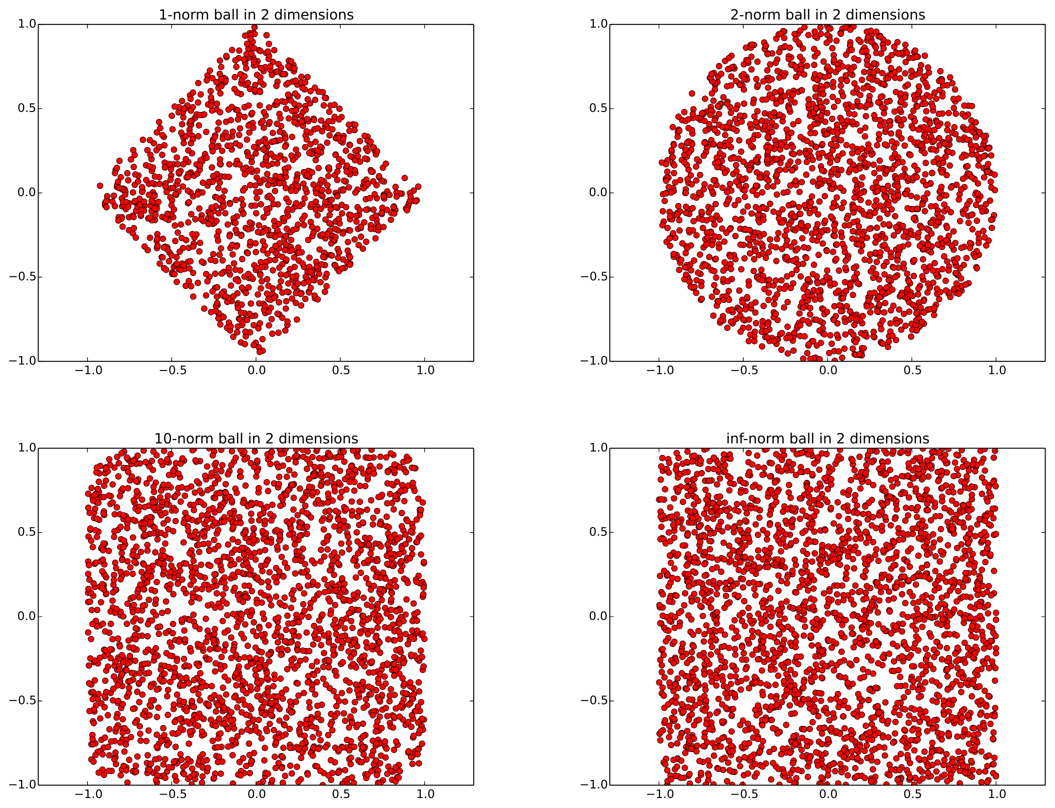


Figure 7: The unit balls with respect to the ℓ^p-norm ($p = 1, 2, 10, \infty$).

(b) We can use the *Monte Carlo* approach to estimating the probability of events, or, equivalently, the volume of a set. The idea is: generate randomly distributed points from a set whose volume you know, that contains the set whose volume you'd like to compute. Then, count the fraction of the random points that land inside your set. Use this approach to approximate the volume of the unit ball with respect to the ℓ^p norm, for $p = 1, 2, \infty$.

Solution See Code 1.

```
import numpy as np
N_SAMPLE = 10**6
DIM = 10
p = 2 # p-Norm ball
# Generate samples uniformly at random from [-2,2]^DIM
samples = np.random.uniform(low=-2.0,high=2.0,size=(N_SAMPLE,DIM))
# Select the samples in the unit p-norm ball
samples_in_ball = samples[np.linalg.norm(samples,p,axis=1)<= 1]
# Monte Carlo to compute the volume
n_sample_in_ball = samples_in_ball.shape[0]
approx_vol = (4.0**DIM)*n_sample_in_ball/N_SAMPLE
print approx_vol
```

Code 1: Estimation of the volume of the unit ball with respect to the ℓ^p norm

(c) (Bonus) Now, try to use this Monte Carlo approach to estimate the probability of the ℓ^p ball, for $p = 2$, in higher dimensions. Try 5, 10 and then 100 dimensions. Do you succeed? Note that you can just look up the answer to check if your Monte Carlo result is correct, or even close to correct. Comment on the success and/or ease of your method.

The moral of this exercise is that while Monte Carlo is a very powerful approach to solve problems by simulation, you have to be careful. There are more sophisticated Monte Carlo techniques that allow us to solve more complicated problems by sampling, and these deal directly with the problems you should have come across as you increased the dimension.

Solution We can note that it is more likely to have an approximate volume with larger error when we play with higher dimensional space if the number of samples are the same.

6. In this last problem, we explore gradient descent, convergence rates, and what's called the *step size* or *learning rate* of an iterative problem.

(a) To warm up, write a gradient descent routine to solve the 1-D least squares regression problem that you wrote down in class on Sunday. Recall that the gradient descent algorithm initializes with some x_0, and then updates via

$$x^{(k+1)} = x^{(k)} - \eta f'(x^{(k)}).$$

Choose $\eta < f''(x)$ and $\eta > f''(x)$, and comment on what you observe, ideally with a graphical explanation of why there is such a big difference between the two cases.

Solution We want to solve the following problem using the gradient descent algorithm:

$$\min_{x \in \mathbb{R}} f(x) := (ax - b)^2.$$

For this problem, the update equation is as follows:

$$x^{(k+1)} = x^{(k)} - 2\eta a(ax^{(k)} - b).$$

We can rewritten this as follows:

$$x^{(k+1)} - \frac{b}{a} = (1-2\eta a^2)(x^{(k)} - \frac{b}{a}) \implies x^{(k)} - \frac{b}{a} = (1-2\eta a^2)^k(x^{(0)} - \frac{b}{a})$$
$$\implies x^{(k)} = \frac{b}{a} + (1-2\eta a^2)^k(x^{(0)} - \frac{b}{a}).$$

Therefore, if $|1-2\eta a^2| < 1 \iff 0 < \eta < \frac{1}{a^2}$, $x^{(k)}$ converges to the optimal point $x^* = b/a$ by the gradient descent algorithm.

(b) Now for the higher dimensional setting. Here, gradient descent takes the form

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \eta\nabla f(\boldsymbol{x}^{(k)}).$$

Here, $\nabla f(\boldsymbol{x})$ denotes the *gradient* of the function f at the point $\boldsymbol{x}$. If f is a function that takes n arguments, $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$, then its gradient is a $n \times 1$ vector:

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{x}) \\ \frac{\partial f}{\partial x_2}(\boldsymbol{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\boldsymbol{x}) \end{pmatrix}.$$

Compute the gradient for the following functions:

- $$f(\boldsymbol{x}) = 2x_1 - x_2 + 5x_3.$$
- $$f(\boldsymbol{x}) = x_1^2 x_2 + x_2 \sin x_3 + 2x_3.$$

Solution

- $\frac{\partial f}{\partial x_1}(\boldsymbol{x}) = 2$, $\frac{\partial f}{\partial x_2}(\boldsymbol{x}) = -1$, $\frac{\partial f}{\partial x_3}(\boldsymbol{x}) = 5$. Therefore,
$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2 \\ -1 \\ 5 \end{bmatrix}.$$
- $\frac{\partial f}{\partial x_1}(\boldsymbol{x}) = 2x_1x_2$, $\frac{\partial f}{\partial x_2}(\boldsymbol{x}) = x_1^2 + \sin x_3$, $\frac{\partial f}{\partial x_3}(\boldsymbol{x}) = x_2 \cos x_3 + 2$. Therefore,
$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2x_1x_2 \\ x_1^2 + \sin x_3 \\ x_2 \cos x_3 + 2 \end{bmatrix}.$$

(c) Have a look at the IPython notebook `2D Quadratic Gradient Descent`. Generate three (3) random examples, and while keeping the example fixed, try to find values of η, the step size, for which gradient descent converges, and for which it diverges.

Solution In IPython notebook `2D Quadratic Gradient Descent`, we solve the following problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} f(\boldsymbol{x}) := \boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{q}^T \boldsymbol{x} + q_0.$$

Suppose we fix the example with $Q = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$. We see that the gradient descent converges if $\eta < 1/9$ for this Q, and diverges otherwise (See Fig. 8).

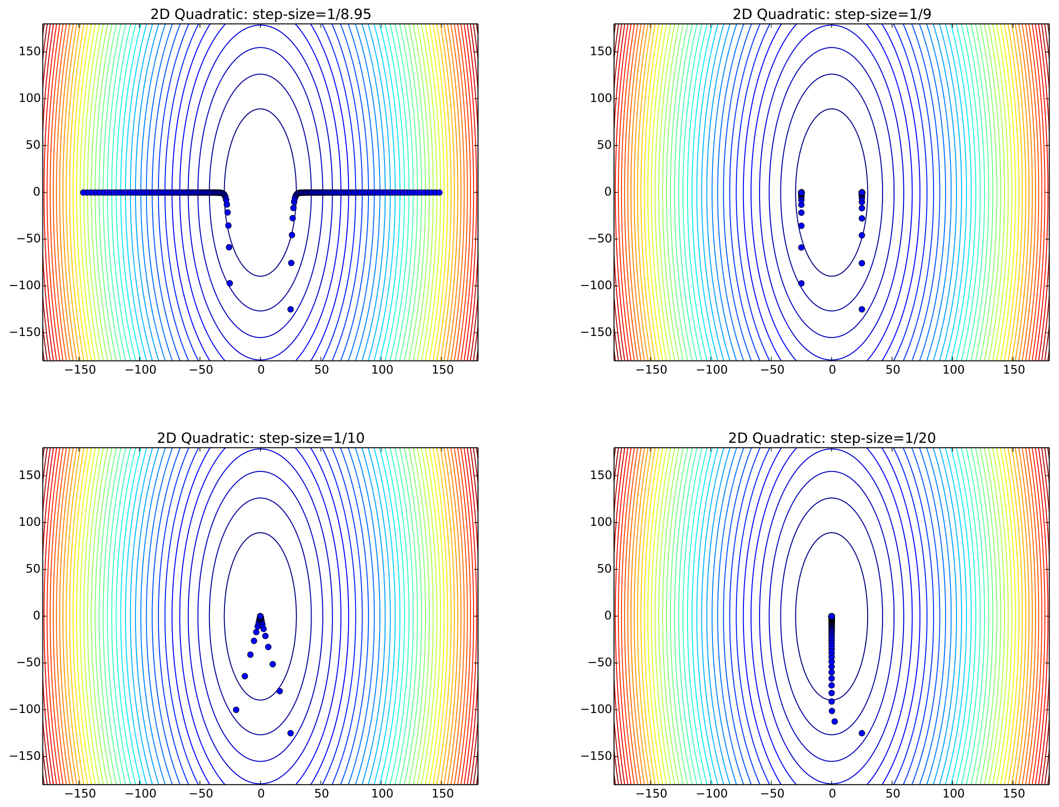


Figure 8: Convergence of the gadient descent algorithm on 2D quadratic objective.

(d) (Commentary... nothing to do...) The key to realize, is that convergence rate depends, just like in the 1 dimensional case, on the curvature of the quadratic. The challenge, conceptually, is that unlike quadratics in 1-D whose curvature is given by the coefficient in front of the squared term, in two dimensions, quadratics do not have just one degree of curvature. Quadratics in two dimensions can be characterized by two curvature numbers. Basically, imagine sitting on a bowl that opens up slowly in one direction, and quickly in the perpendicular direction. It turns out that one need only consider two directions in two dimensions. The right directions are the major and minor axes of the ellipses that make up the *level sets* of the function.

In the IPython notebook near the top, you'll see commands for plotting the level sets, and you will notice that in every single case, you will get ellipsoidal level sets.

So, how do we find these directions, and the steepness in each?

(e) Go back to your work in the second part of this problem, where you found the threshold value of η, where above that gradient descent diverges, and below it converges. Now use the `numpy.linalg.eig` command to find the eigenvalues and eigenvectors of the matrix Q. The command will return two numbers – these are the two eigenvalues of Q – and then two 2-dimensional vectors – these are the eigenvectors of Q. How do the eigenvalues compare to the threshold values of η that you found above?

Solution For the chosen $Q = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$, we have two eigenvalues $\lambda_{max} = 9$, $\lambda_{min} = 1$. We see that the threshold value of η is $1/\lambda_{max}$.

(f) Playing with eigenvalues and eigenvectors: if the eig command returns eigenvalues λ_1 and λ_2, and eigenvectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, compute $Q\boldsymbol{v}_1$ and $Q\boldsymbol{v}_2$. Notice anything? The relationship you notice is precisely the defining equation of eigenvalues and eigenvectors.

Solution We have $Q\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ for $i = 1, 2$.

(g) (Bonus) The above discussion and exercises show that the *largest eigenvalue* governs how large our step size should be. The larger the eigenvalue, the more the curvature and hence the smaller the stepsize. What role does the smallest eigenvalue play? It turns out that the ratio of largest to smallest, controls how quickly we converge.

Generate examples where the largest eigenvalue is fixed (and hence you are using the same stepsize), and vary the second eigenvalue from very small (close to zero, but positive), to very big – almost as big as the largest eigenvalue. Plot the convergence of gradient descent. Can you discern a relationship?

Solution As shown in Fig. 9, the gradient descent algorithm converges in the smaller number of steps for the problem with Q that has a larger smallest eigenvalue.

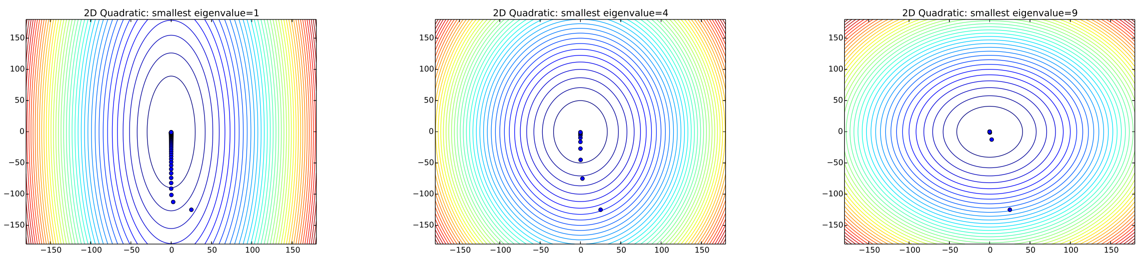


Figure 9: Convergence of the gadient descent algorithm with different curvatures.