# INFO371 Problem set 2

Your name:

Deadline: Wed, Apr 18, 5:30pm

## Introduction

This homework is pretty long. But don't get desperate just yet, it largely repeats related questions, you have to modify your code just a little bit. We haven't talked about gradient and gradient ascent in the class, but take a look at the lecture notes (canvas/files/readings) and the example notebook (canvas/-files/assignments). This should give you good enough background.

Please submit a) your code (notebooks, rmd, whatever) and b) the lab in a final output form (html or pdf). Unlike PS1, you are free to choose either R or python for solving this problem set.

While some of the intermediate output may be informative, please don't include too many lines of it in your solutions!

Note: it includes questions you may want to answer on paper instead of computer. You are welcome to do it but please include the result as an image into your final file.

Working together is fun and useful but you have to submit your own work. Discussing the solutions and problems with your classmates is all right but do not copy-paste their solution! Please list all your collaborators below:

1.
2. ...

## 1 Gradient Ascent

Before you start, please consult the lecture notes about gradient ascent.

### 1.1 Implement The 1D GA Algorithm

As a warm-up exercise, your first task is to implement 1-dimensional version of GA. Let's look at 1D case with $f(x) = -x^2$. Pick an $x^0$ (but don't pick $x^0 = 0$), and set the learning rate $R = 0.1$.

1. Start slow and manual.
   (a) What is the correct location of the maximum of this function?
   (b) What is the gradient vector of the function? What is it's dimension?
   (c) Compute the gradient at $x^0$, $\nabla f(x^0)$.
   (d) Compute $x^1 = x^0 + R \cdot \nabla f(x^0)$.
   (e) Did we move closer to the maximum?
2. Now repeat the previous exercise on computer. Choose at least one stopping criterion and it's parameter $\epsilon$, and the bail-out number of iterations, $R$. Implement the above as an algorithm that at each step prints out the $x$ value, the function's value $f(x)$ and gradient $\nabla f(x)$. At the end it should print the solution, and also how many iterations it took for the loop to converge (to finish).

3. Experiment with different starting values, learning rates and stopping parameter values. Comment and explain your findings.

Congrats! You have implemented the 1D gradient ascent! This was easy, right?

### 1.2 Implement The 2D Version

But now we get more serious: we take a 2D quadratic function From now on, we do everything in matrix form because this scales–there is little difference between 2D and 200D problem if your code is written in matrices.

In matrix form the quadratic function can be expressed as

$$f(\mathbf{x}) = -\mathbf{x}'A\mathbf{x} \tag{0.1}$$

where $\mathbf{x} = (x_1, x_2)'$ is a $2 \times 1$ matrix and $A$ is a $2 \times 2$ matrix. We only look at cases where $A$ is symmetric.

4. Write $\mathbf{x} = (x_1, x_2)'$ and $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$. Show that $f(\mathbf{x}) = -\mathbf{x}'A\mathbf{x}$ describes a quadratic function.

   To show this, I recommend to write the function in non-matrix form as a formula explicitly in the components of $\mathbf{x}$, $x_1$ and $x_2$.

5. What is the condition that the problem has a single maximum, i.e. $f(\mathbf{x}) \leqslant 0$ for all $\mathbf{x}$? What's the location of it's true maximum?

   Hint: Consult Greene Appedix A7, p834 (available in files/readings).

6. Compute the gradient vector of this function. Use the non-vector form you did under 4 above.

   Note: you may not be familiar with vector derivatives, but in vector form

$$\nabla f(\mathbf{x}) = -(A' + A)\mathbf{x} \quad \text{or} \quad -2A\mathbf{x} \qquad \text{as } A \text{ is symmetric.} \tag{0.2}$$

   Show that this is indeed true for the 2D case by taking the partial dericatives from the formula you computed in 4 above.

7. Now code the algorithm above for 2D case. Do it in matrix form! Show that you get the correct solution if you pick $x^0 = (2, -3)'$ and $A = \begin{pmatrix} 1 & 2 \\ 2 & 8 \end{pmatrix}$.

   As before, experiment with a few different learning rates and see how does this influence the speed of convergence. Comment and explain your findings.

8. Visualize your algorithm's work:

   Hint: look at the example notebook code.

   - mark all the intermittend points (the $\mathbf{x}^1$-s) on the plot and connect these with lines.

     Note: you may want to save, instead of print, these values in your code now. In terms of showing the points on the figure, I recommend to plot the 100 first points, and the last point.
   - add the function contours on the plot. In R you can use `graphics::contour` or `ggplot2::geom_contour`. In matplotlib `plt.contour` does a similar job.

     Note: try to use fixed 1:1 aspect ratio. Otherwise the direction of gradient does not coincide with that of the steepest ascent on the figure.
   - Explain what you see. A correctly working GA algorithm's path is shown on Figure 1. But note that depending on your parameters, the picture may look different.

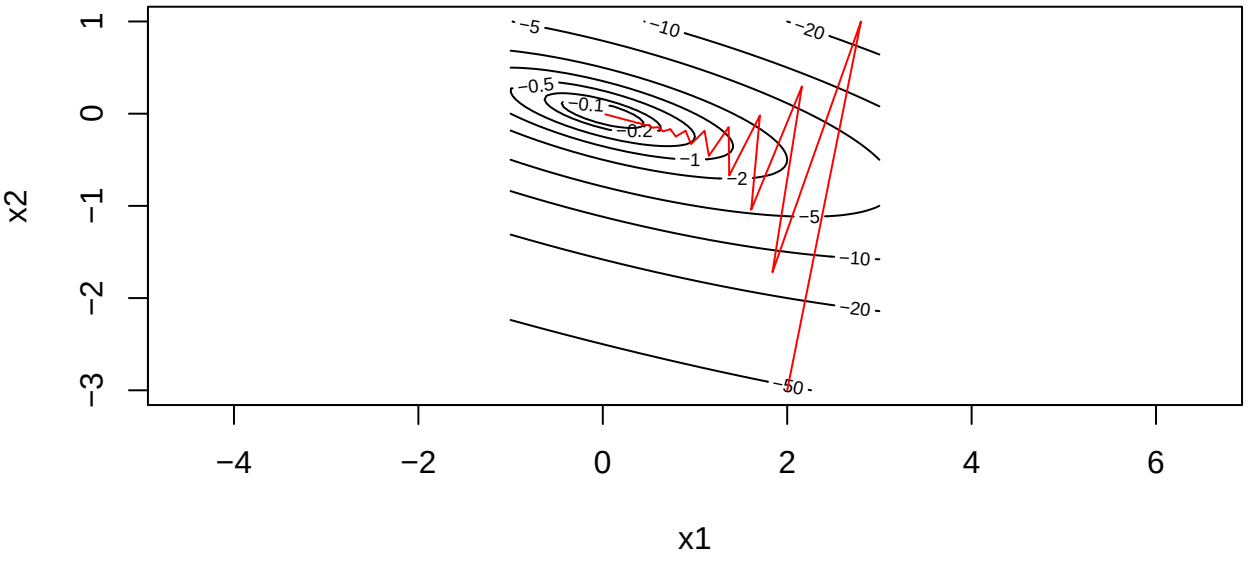


Figure 1: Example path of GA convergence

9. Finally, let's try if your code scales to 5D case without any modifications. Take

$$\mathsf{A} = \begin{pmatrix} 11 & 4 & 7 & 10 & 13 \\ 4 & 17 & 10 & 13 & 16 \\ 7 & 10 & 23 & 16 & 19 \\ 10 & 13 & 16 & 29 & 22 \\ 13 & 16 & 19 & 22 & 35 \end{pmatrix} \tag{0.3}$$

(generated as $\frac{1}{2}(\mathsf{A}+\mathsf{A}')+10\cdot\mathsf{I}_5$ where $\mathsf{A}$ is a $5\times5$ matrix of sequence $1\ldots25$, and $\mathsf{I}_5$ is the corresponding unit matrix.) Take the initial value $\mathbf{x^0} = \mathsf{c}(10,20,30,40,50)'$. Play with hyperparameters until you achieve convergence! Show and comment your results. In particular I'd like to see your comments comparing the easiness and speed of getting the 1D, 2D and 5D case to converge.

### 1.3 Condition Numbers

How easy it is to get your algorithm to converge also depends on how close or far is your matrix from being singular. This can be done using *condition number*.

10. Compute the condition number for matrix $\mathsf{A}$ in the example above.

    Hint: consult Greene, Appendix 6.6, p 829.

    Note: a) Greene uses a slightly less common definiton of condition number, typically one uses just the fraction, not square root of it as Greene does. b) the words "largest" and "smallest" should be understood as largest and smallest *in absolute value*.

Now let's return to 2D case. Take a simple singular matrix $\mathsf{A} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

11. Show, on computer, that it is singular! Show it without error messages but using relevant linear algebra functions/properties instead.

Let's a non-singular matrix $\mathsf{C}$ instead by adding a "certain amount" $\alpha$ of unit matrix to it: $\mathsf{C} = \mathsf{A} + \alpha \cdot \mathsf{I}_2$. Your task is to analyze and visualize the algorithm's work for three cases: $\alpha \in 100, 1, 0.01$.

12. Run your algorithm for each of these three cases. For each $\alpha$:
    (a) report the hyperparameters
    (b) compute the corresponding $\mathsf{C}$
    (c) print it's eigenvalues and condition number. Please state which definition of eigenvalues you are using if not following Greene!
    (d) solve the problem using GA
    (e) report the location of maximum and the number of iterations it took
    (f) visualize the process using a figure, similar to the 8 above. Try to play with the 'level' argument of the contour plot to make the shape of the resulting object clear.

    Note: you may have to set different hyperparameters for different $\alpha$-s.

13. Comment on your results. How is the convergence speed related to the matrix condition number?

## 2 Outliers in different distributions

The task in this problem is similar to lab 1–conduct a series of MC simulations and see how often do we get outliers of given size. How often do we get "statistically significant" results even if there is nothing significant in our model.

You will generate a large number $\mathsf{R}$ (1000 is a good choice) of random samples of given size $\mathsf{N}$ (with $\mathsf{N}$ between 10 and 10 millions), and take mean of these samples. What is the distribution of the means? How does the distribution depend on the sample size $\mathsf{N}$? And what's difference if the random numbers are drawn from different samples?

### 2.1 Normal Distribution

Here we essentially repeat the lab.

1. Pick your number of repetitions $\mathsf{R}$ (1000 is a good choice), and sample sizes $\mathsf{N}$ (10, 1000, 100,000 are good choices, but you may want to adjust if the latter is too slow).

2. For $\mathsf{R}$ times create sample of $\mathsf{N}$ of standard normals, and take the mean of it. You'll have $\mathsf{R}$ means of $\mathsf{N}$ standard normals.

3. Compute the mean-of-means, it's standard deviation, and 95% confidence region.

   The 95% confidence region is the opposite of computing the percentage of observations that fit into the interval. The easiest way to find it is using sample quantiles. Check out the `quantile` function in R, or `np.percentile` function in python. For instance, `np.percentile(x, 10)` will give the 10th percentile $\mathsf{q}_{10}$ of an array $\mathsf{x}$. 10th percentile is such a value that 10% of the sample values are smaller than that value.

4. Repeat the above with three different (and very different) $\mathsf{N}$-s, such as 10, 1000, 100,000. Show how the mean-of-means, it's standard deviation, and the confidence region depends on $\mathsf{N}$.

## 2.2 Pareto distribution

Normal distribution has very nice properties, despite it's density function looking a little bit ... nasty.

But there are many things in our world that are not well approximated by normal distribution. Examples include size of cities, links to webpages, popularity of actors, number of citations of researchers, size of wildfires.... All of these are very inequal–there are cities that are enourmous, but most of towns are small. Some researchers have hundreds of thousands of citations but most of them only a few. This kind of outcomes can be described by *Pareto distribution*. There are many ways to define it, but let's do it like this:

$$F(x) = 1 - \left(\frac{x}{x_0}\right)^{-k} \quad \text{and} \quad f(x) = k x_0^k x^{-(k+1)} \qquad \text{where } x \geqslant x_0 \text{ and } k > 0. \tag{0.1}$$

The parameter $x_0$ (called *location*) tells where the distribution "begins" and $k$ describes how fast it falls for large $x$ values (called *shape*). The expected value for Pareto random variable is

$$\mathbb{E}\, X = \frac{k}{k-1} x_0 \qquad \text{for } k > 1. \tag{0.2}$$

Note that for $k \leqslant 1$ the expected value does not exist! Your task is to explore what the property means in practice.

Set $x_0 = 1$ and pick a $k \leqslant 1$. Note: depending on the software you use, it may parameterized differently and you may have to adjust the parameter choice accordingly. If you use `np.random.pareto` (python) or `VGAM::rpareto` (R), the parameterization is as described here.

5. Create a large number of pareto random variables. Show their distribution on a histogram. Attempt to make the histogram look good.

   Note: you'll see it is hard to make it to look good. But you may try to do it in log-log scale, and set an upper limit to the variables you will display, say only values $x < 100$ on the histogram.

Now repeat exactly the same task as you did above with random normals:

6. Pick $R$ and three $N$ values
7. Create $R$ times a sample of $N$ Pareto values, and take a mean of those.
8. Compute the mean-of-the-means, it's standard deviation, and 95% confidence region.
9. Repeat it with 3 different $N$-s.
10. Finally compare your results for normal and Pareto distributions.