DAMSL-091 - Numerical Algorithms
Final Project

Due: 18:00 14/01/2025,
Presentation 09:00 16/1/2025 at the lab, B203

1 The PageRank algorithm

1.1 Introduction

The *PageRank* algorithm is the core of the Google search engine. It estimates the importance of a page in the world wide web(www). It was proposed by S.Brin and L. Page in 1998 in the paper entitled *The anatomy of a large-scale hypertextual Web search engine*, Computer Networks and ISDN Systems. 30 (1–7): 107–117. As described in the paper, the algorithm is defined as follows:

> We assume page A has pages $T_1, \ldots, T_n$ which point to it (i.e., are citations). The parameter d is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C(A) is defined as the number of links going out of page A. The PageRank of a page A is given as follows:
>
> $$PR(A) = (1-d) + d\Big(PR(T_1)/C(T_1) + \cdots + PR(T_n)/C(T_n)\Big)$$
>
> Note that the PageRanks(PR) form a probability distribution over web pages, so the sum of all web pages' PageRanks will be one. PageRank or PR(A) can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web.

The normalized matrix, the algorithm refers to, is a *Markov* type matrix. These are square matrices with elements that express probabilities and the elements of each column add up to 1. The largest eigenvalue of such matrix is 1 with multiplicity 1 and the elements of the corresponding eigenvector add up to 1. The rest of the eigenvalues are, in absolute value, smaller than 1.

We assume that we have N webpages with multiple connections between them. Then, the related *Markov* type matrix has the following form

$$M = d \cdot A + \frac{1-d}{N} B, \quad B = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \end{bmatrix}$$

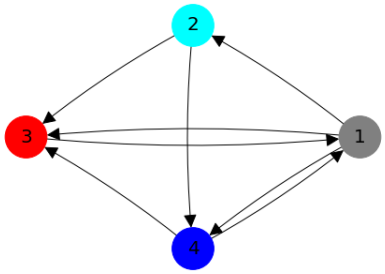
where $d \sim 0.85$ and A is a matrix with elements a_{ij}

$$a_{ij} = \begin{cases} \frac{1}{L(j)} & \text{if there is a link from page } j \text{ to page } i \\ 0 & \text{otherwise} \end{cases}$$

where $L(j)$ is the number of all outgoing links from page j.

1.2 Implementation

An an example, consider the following directed graph where each node of the graph represnts a webpage with the corresponding links



and $L(1) = 3,\ L(2) = 2,\ L(3) = 1,\ L(4) = 2$. In this case, the matrix A is

$$A = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

A rather fast way to compute the dominant eigenvalue λ of a matrix $A \in \mathbb{R}^{n,n}$ and the corresponding eigenvector x is the *Power method*, which is an iterative algorithm which produces successive approximations x_k to the eigenvector x and using the Rayleigh quotient provides an estimate for the dominant eigenvalue:

Power method for $Ax = \lambda x$

Choose a random $x_0 \in \mathbb{R}^n$, $x_0 = x_0/\|x_0\|$
Choose k_{max} - maximum number of iterations
Choose $\epsilon <<< 1$ a small tolerance
$k = 0,\ d_k = 1$
While $d_k > \epsilon$ and $k < k_{max}$ do
    $x_k = Ax_{k-1}$
    $x_k = x_k/\|x_k\|$
    $d_k = \|x_k - x_{k-1}\|$
$\lambda = \frac{x_k^T A x_k}{x_k^T x_k}$

Write a code in Python that implements the power method, then reads a graph from a file, constructs matrix M and computes its dominant eigenvalue and eigenvector. We take as initial vector $x_{0,i} = \frac{1}{N},\ i = 1, \ldots, N$. As a $\|\cdot\|$ you can use one of $\|\cdot\|_1,\ \|\cdot\|_2,\ \|\cdot\|_\infty$. Each component of x_k represents the importance(rank) of the corresponding node(page) of the graph(web). Test your code with the graphs in graph0.txt, graph1.txt, graph2.txt and make sure it works by confirming that $\lambda \approx 1,\ \sum_{i=1}^{N} |x_{k,i}| \approx 1$. At the end your code will list the nodes(pages) in descending order with respect to their importance(rank).

The graph(web) is stored in a file, where each line is of the form $n_i\ n_j$ which means that there is a link from node(page) n_i to node(page) n_j. For example the file `graph0.txt` contains

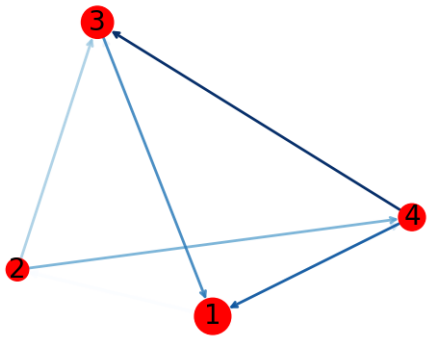
the description of the simple graph above and is

1 2
1 3
1 4
2 3
2 4
3 1
4 1
4 3

1.3 The `NetworkX` Python library

A simple way to construct, process and visualize graphs in Python is with the library *NetworkX*. The library has also an implementation of the PageRank algorithm. You can install the library in your Python environment and then use its classes to construct a directed graph `G`, using as input one of the files graph0.txt, graph1.txt, graph2.txt. Then, use the PageRank algorithm `networkx.pagerank(G, d)` and compared it with your results obtained by the power method.

Further, we can visualize the graph with the size of its node(page) is proportional to its importance(rank).



2 Function Minimization

2.1 Introduction

We consider the problem of finding $x^* \in \mathbb{R}^n$ which minimizes a function $f : \mathbb{R}^n \to \mathbb{R}$:

$$x^* = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} f(x)$$

This problem can be solved analytically in rare cases. Then, one relies on solving the problem numerically by approximating x^* using the *Gradient Descent(GD)* algorithm:

Gradient Descent(GD)
Choose a random $x_0 \in \mathbb{R}^n$ For $k = 0, 1, \ldots$ do $x_{k+1} = x_k - \alpha \nabla f(x_k)$

where α is the step(or learning rate) of the method. In the literature there are several variations of the GD algorithm that include choosing appropriate step α, ways to evaluate the gradient of f efficiently, accelerate the convergence, and reduce the computational cost.

2.2 Python Implementation

For a given function $f : \mathbb{R}^n \to \mathbb{R}$ with derivative ∇f write a Python code that implements the GD algorithm and accepts the following parameters: k_{max} the maximum allowed number of iterations, α the step of the method, ε an accuracy parameter. The iterative process will stop if one of the following criteria is satisfied: a) $k > k_{max}$, b) $\|x_{k+1} - x_k\| \leq \varepsilon$, c) $\|\nabla f(x_k)\| \leq \varepsilon$ displaying an appropriate message. At the end your code will display the number of iterations k, and the last value of x_k and $f(x_k)$. Also in a graph the objective function f will be displayed along with the path, i.e the sequence of points $(x_k, f(x_k))$, the algorithm followed to reach the minimum, for example see Figure 2.

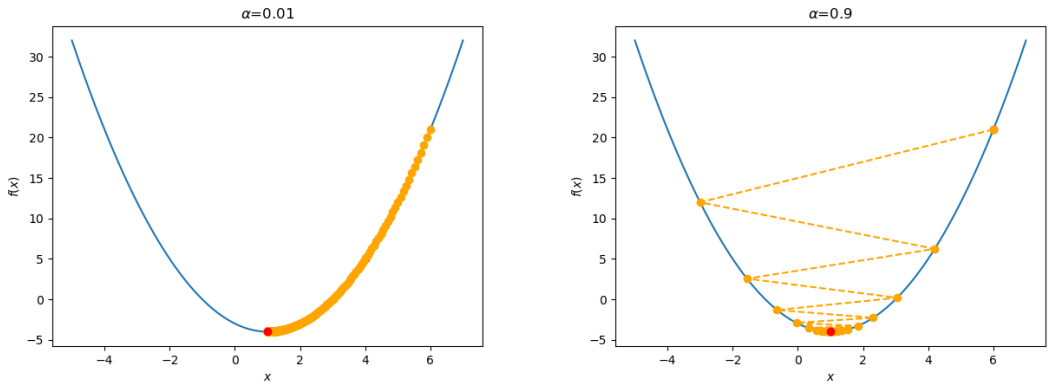


Figure 1: Sequence of points(path) in the GD algorithm for a 1D objective function

In a different graph the number of iterations versus the value of the objective function(in logarithmic scale) will be displayed, for example

Test your code for the following objective functions

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2, \quad \text{Rosenbrock, Global Minimum : } f(1,1) = 0$$

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48 x_1 x_2, \quad \text{Matyas, Global Minimum : } f(0,0) = 0$$

$$f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n}(x_i)^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right), \; n = 1, 2 \text{ Griewank, (many local minima)}$$

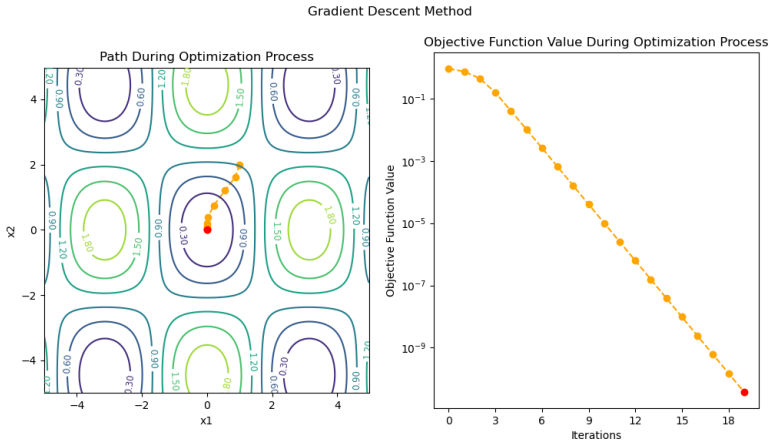


Figure 2: Sequence of points(path) in the GD algorithm reduction graph for a 2D objective function

2.3 Backtracking algorithm

An important ingredient of the GD algorithm is the stepsize parameter α. The usual choice is to select a fixed value α which has its advantages and disadvantages. In the literature there are many different algorithms to choose α adaptively which affects positively the robustness of the method and reduces its computational cost by accelerating the convergence. One such algorithm is the Backtracking Algorithm. The main idea of the algorithm is to reduce the size of α progressively (i.e. backtracking) until a desired reduction of the objective function is achieved. The abstract form of the algorithm is :

Backtracking
Choose a α_0
Choose $\tau \in (0,1)$ and $c \in (0,1)$
Set $m = \nabla f(x)^T p$
Set $j = 0,\ t = cm$
While $f(x + \alpha_j p) > f(x) + \alpha_j t$ do
$\quad \alpha_{j+1} = \tau \alpha_j$
$\quad j = j + 1$

where the direction of reduction is $p = -\nabla f(x)$.

Implement in Python a version of the GD algorithm that uses at every iteration the backtracking algorithm for choosing α. Apply the new algorithm to the aforementioned functions and compare its results to the standard GD method. Display the same output as before and determine any advantages or disadvantages.

3 An approach for the face recognition problem

Traditionally, face recognition has ignored the issue of just what aspects of a face stimulus are important for identification. An information theory approach of coding and decoding face images may give insight into the information content of face images, perhaps different from our intuitive notion of face features such as eyes, nose, lips and hair. In mathematical terms, we wish to find the principal components of the distribution of faces, or the eigenvectors of

the covariance matrix of the set of face images, treating an image as a point (or vector) in a very high dimensional space. The eigenvectors are ordered, each one accounting for a different amount of the variation among the face images. These eigenvectors can be thought of as a set of features that together characterize the variation between face images. Each image location contributes more or less to each eigenvector, so that we can display the eigenvector as a sort of ghostly face which we call an eigenface. Each individual face can be represented exactly in terms of a linear combination of the eigenfaces. Each face can also be approximated using only the "best" eigenfaces—those that have the largest eigenvalues, and which therefore account for the most variance within the set of face images. The best n eigenfaces span an n-dimensional subspace, the "face space" of all possible images.

Assume that each face has $m \times n = M$ pixels and is represented as an M-vector. Given N face images $\mathbf{f}_i$, $i = 1, \ldots, N$, of known individuals, we form the $M \times N$ matrix

$$S = [\mathbf{f}_1, \mathbf{f}_2, \ldots f_N], \tag{1}$$

and the mean image

$$\bar{\mathbf{f}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{f}_i. \tag{2}$$

Subtracting $\bar{\mathbf{f}}$ from each face image gives

$$\mathbf{a}_i = \mathbf{f}_i - \bar{\mathbf{f}}, \quad i = 1, \ldots, N, \tag{3}$$

and another $M \times N$ matrix

$$A = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_N]. \tag{4}$$

We let $r = \text{rank}(A) \leq N$ and consider the singular value decomposition of the matrix A

$$A = U \Sigma V^T, \tag{5}$$

where Σ is an $M \times N$ diagonal matrix of singular values

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_N = 0, \tag{6}$$

and $U \in \mathbb{R}^{M \times M}$, $V \in \mathbb{R}^{N \times N}$ are orthonormal matrices. The vectors $\mathbf{u}_i$, $i = 1, \ldots, r$, form an orthonormal basis of $\mathcal{R}(A)$, the range of A. We think of $\mathcal{R}(A)$ as a "face subspace" in the image space consisting of $m \times n$ images. Now let

$$\mathbf{x} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r]^T (\mathbf{f} - \bar{\mathbf{f}}), \qquad \mathbf{x} = [x_1, x_2, \ldots x_r]^T, \tag{7}$$

be the coordinates of the scalar projection of $\mathbf{f} - \bar{\mathbf{f}}$ onto the face subspace. The vector $\mathbf{x}$ is used to find which of the faces corresponding to known individuals best describes the face $\mathbf{f}$: we find some face $\mathbf{f}_i$, $i = 1, 2, \ldots, N$, that minimizes the distance

$$\epsilon_i = \|\mathbf{x} - \mathbf{x}_i\|_2, \quad i = 1, 2, \ldots, N, \tag{8}$$

where $\mathbf{x}_i$ is the projection of $\mathbf{f}_i - \bar{\mathbf{f}}$ onto the face subspece, that is,

$$\mathbf{x}_i = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r]^T (\mathbf{f}_i - \bar{\mathbf{f}}). \tag{9}$$

A face $\mathbf{f}$ is classified as face $\mathbf{f}_i$ if the distance ϵ_i is less than some threshold ϵ_0. Otherwise the face is classified as "unknown".

Now if $\mathbf{f}$ is not a face, let $\mathbf{f}_p = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r]\, \mathbf{x}$, with $\mathbf{x}$ given from (7). It's distance from the face subspace will be nonzero and equal to the distance of $\mathbf{f} - \bar{\mathbf{f}}$ and $\mathbf{f}_p$

$$\epsilon_f = \|(\mathbf{f} - \bar{\mathbf{f}}) - \mathbf{f}_p\|_2. \tag{10}$$

Thus, if ϵ_f is greater than some threshold ϵ_1 then $\mathbf{f}$ is not a face. We summarize the steps involved as follows:

1. Suppose that a set S of N known faces is given.
2. We compute the mean face $\bar{\mathbf{f}}$ and the matrix A from (4).
3. We compute the singular value decomposition of A.
4. For each face $\mathbf{f}_i$ we compute the vectors $\mathbf{x}_i$ from (9)
5. For a new face $\mathbf{f}$ to be identified calculate $\mathbf{x}$ from (7), the projection $\mathbf{f}_p$ and the distance ϵ_f. If $\epsilon_f > \epsilon_1$ then the imput is not a face
6. Otherwise, compute the distances ϵ_i from (8). If $\epsilon_i > \epsilon_0$ for all $i = 1, 2, \ldots, N$, then the image be classified as an unknown face. If $\epsilon_i < \epsilon_0$ for some i then the image be classified as the known individual associated with the coordinate vector $\mathbf{x}_i$ in (8).

Regarding the implementation, the folder `faces` contains $N = 36$ JPEG files each of size 320×320 pixels resulting in $M = 102400$. We read each image file and place it as a column in the matrix S. We compute the mean image and normalize images by subtracting the mean. This builds the matrix A. We compute its singular value decomposition and the coordinate vector $\mathbf{x}_i$ for each face. To test the method we use the images in the folder `testfaces`. The JPEG files with a numerical base name, like `11.jpg` are faces also appearing in the folder `faces`. The image `notface.jpg` is not a face, while `U1.jpg`, `U2.jpg` and U3.jpg are faces that do not appear in the folder `faces`. Their dimensions are again 320×320. To classify each of these images we normalize it subtracting the mean image we computed earlier, we compute its coordinate vectori $\mathbf{x}$ and its distance ϵ_f from the face subspace. If it's distance is greater then ϵ_1 then the image is not a face. Otherwise, we compute the distances ϵ_i and pick the smallest one, provided it is smaller than ϵ_0. Otherwise, it belongs to an unknown person. Experiment with the values of the threshold. Use $\epsilon_1 < \epsilon_0$. Given the dimension of the image space both thresholds should be rather high, say around 10 or more.

Some hints about the implementation:

- Read each of the N images into the columns of some array S. If the dimensions of the image is $M = m \times n$ then S should have M rows. The `imread` will be useful here.
- Find the mean image, subtract from each column and compute the SVD of the resulting matrix
- Compute $\mathbf{x}$ from (7). The rank of the matrix A is the number of its columns
- Given some image to classify, find the corresponfing coefficient vector $\mathbf{x}$ from (7) and compute ϵ_f
- Compare ϵ_f with threshold ϵ_1. If ϵ_f is smaller, compute the ϵ_i from (8). This is just the inner product of the i-th column of $\mathbf{x}_i$ minus the vector $\mathbf{x}$, with itself. The the k-th threshold ϵ_k is the smallest, then the image is that of the k-th face, otherwise we came across a new face.

4 Instructions for submission

For this project you can work individually or in teams of up to two people. Please write a report in LaTeX in which you will include all the results, tables, graphs etc with necessary comments and explanations. The Python programmes will be submitted through the course webpage in UoC-eLearn. You will submit a single file in zip format named Project.zip which will contain:

a) the report(tex and pdf), b) all the codes. Each Python code must have the following name structure : Project_PX.py with X is the number of the corresponding problem. At the top of each file, as a comment, there will be your name(s) and student ID(s).

The project presentation will be on the date of the final exam. Each team, will prepare a presentation of the project and present it to the class. The total time is 30 minutes, which includes 20-25 minutes of presentation 5-10 minutes of questions/comments.

All problems carry the same grade points with max being 100. Discussions with fellow students is encouraged, however you need to submit your own work. Solutions or codes that are the result of plagiarism will get no marks.