ETH zürich

High Performance Computing for Science and Engineering I

J. H. Walther, S. Martin

Fall semester 2022

Set 3 - Finite Differences & PCA

Issued: October 26, 2022
Hand in (optional): November 9, 2022, 08:00

Question 1: Diffusion (50 points)

The diffusion of a substance can be described by the equation

$$\frac{\partial c(x,y,t)}{\partial t} = D\left(\frac{\partial^2 c(x,y,t)}{\partial x^2} + \frac{\partial^2 c(x,y,t)}{\partial y^2}\right), \tag{1}$$

where c is the concentration of the substance at position $(x,\ y)$ and at time t, and D is the diffusion constant. The diffusion process happens in the domain $|x| < L/2$ and $|y| < L/2$. The concentration is zero on the boundaries of the domain for $t \geq 0$. The initial concentration is

$$c(x,y,0) = \begin{cases} 1, & \text{if } |x| < L/4 \text{ and } |y| < L/4, \\ 0, & \text{otherwise.} \end{cases}$$

a) Write down the 2-dimensional discretized diffusion process for eq. (1). Assume a uniform grid with spacing h and a central finite difference scheme in space and forward Euler time integration. For the forward Euler integration assume time intervals of size dt. Make sure that you annotate all variables.

The corresponding discretized equation is:

$$c_{i,j}^{t+1} = c_{i,j}^{t} + D\frac{dt}{h^2}(c_{i+1,j}^{t} + c_{i-1,j}^{t} + c_{i,j+1}^{t} + c_{i,j-1}^{t} - 4c_{i,j}^{t}) \tag{2}$$

where $c_{i,j}^t = c(i*h, j*h, t)$ is the concentration at time t for $x = i*h$ and $y = j*h$.

Grading scheme:

- 5p: Correct final equation
- -1p: If annotations missing (e.g. $c_{i,j}^t = c(i*h, j*h, t)$)

Total: 5p

In the following subquestions, you will work with the codes provided in `/hw04/code_q1/skeleton_code/`. Please have a look at the README file for further information. We recommend compiling and running the code on the Euler cluster `https://scicomp.ethz.ch/wiki/Main_Page`.

b) Based on the discretization found in the previous subquestion, find the maximal timestep dt using the von Neumann stability analysis. Replace the hardcoded timestep ($t = 0.0001$) in the code with your solution.

We assume a solution of the form $c_{i,j,t} = \rho^t e^{ikx_i} e^{ikx_j} = \rho^t e^{ik(x_i+x_j)}$ and apply this to the discretization

$$\begin{aligned}\rho^{t+1} e^{ik(x_i+x_j)} &= \rho^t e^{ik(x_i+x_j)} + D\frac{dt}{h^2}\rho^t(e^{ik(x_i+h+x_j)} + e^{ik(x_i-h+x_j)} + e^{ik(x_i+x_j+h)} + e^{ik(x_i+x_j-h)} - 4e^{ik(x_i+x_j)}) \\ &= \rho^t e^{ik(x_i+x_j)} + D\frac{dt}{h^2}\rho^t(2e^{ik(x_i+h+x_j)} + 2e^{ik(x_i-h+x_j)} - 4e^{ik(x_i+x_j)})\ .\end{aligned}$$

Dividing both sides by $\rho^t e^{ik(x_i+x_j)}$ gives

$$\rho = 1 + \frac{Ddt}{h^2}(4\cos(kh) - 4)\ .$$

For stability we must have $|\rho| \leq 1$. This yields the upper bound

$$\rho_{max} = \frac{h^2}{4D}$$

The implementation can be found in `homeworks/hw03/code_q1/diffusion.cpp`.

Grading scheme:

- 2p: Correct initial assumption and replacement.
- 1p: Solving for ρ.
- 1p: Finding ρ_{max}.
- 1p: Implementation.

Total: 5p

c) Based on the discretization found in the first subquestion, provide a cache-friendly implementation of the diffusion equation in the method `advance`. I.e. avoid copying of memory if possible and mind the access patterns of the memory. Blocking must not be implemented.

The implementation can be found in `homeworks/hw03/code_q1/diffusion.cpp`.

Grading scheme:
Inside `advance`:

- 10p: Correct implementation of diffusion
- 3p: Efficient memory access scheme inside for loops in order to benefit from data locality.
- 2p: Avoid copying memory of `c_tmp` into `c` but instead use `std::swap` or similar.

Total: 15p

d) Plot the total concentration as a function of time for $t \in [0, 0.5]$ using $D = 1$, $L = 2$ and $N = 100$. The concentration can be read from the file `diagnostics.dat` (column 0 and 1). Qualitatively explain the behaviour of the graph in less than 3 sentences, is this result expected?

An exemplary plotting script can be found in `homeworks/hw03/code_q1/plotdiagnostics.py`. The total concentration decreases over time because of the applied, "absorbing", boundary conditions (concentration outside the domain is zero). The concentration diffuses from high (in the middle of the domain) to low (boundaries of the domain), and once it reaches the boundary it becomes "absorbed", i.e. exits the domain of interest.

Grading scheme:
Plotting:

- 3p: Correct behavior of graph, graph visible for $t \in [0, 0.5]$, x-axis and y-axis labelled, x and y-axis ticks visible and value range clear.
- -1p: Deduct one point if plot not complete (e.g. range is cut-off, or labels missing, or value range not clear).

Explanation:

- 2p: For correct argument.

Total: 5p

e) Parallelize the diffusion process (your implementation from subquestion 1c) in the method `advance` using OpenMP.

The implementation can be found in `homeworks/hw03/code_q1/diffusion.cpp`.

Grading scheme:
Inside `advance`:

- 5p: `#pragma omp parallel for` around outer loop (or similar). Consider cache-friendliness of your parallelization approach.

Total: 5p

f) Parallelize the integration of the concentration (marked with `TODO` in `compute_diagnostics`) and the calculation of the histogram (marked with `TODO` in the method `compute_histogram`) using OpenMP.

The implementation can be found in `homeworks/hw03/code_q1/diffusion.cpp`.

Grading scheme:
Inside `compute_diagnostics`:

- 5p: Correct reduction of variable `amount` (e.g. `#pragma omp parallel for reduction(+:amount)`, or manual reduction with critical section, or similar).

Inside `compute_histogram`:

- 5p: Correct reduction of variables `max_c` and `min_c` (e.g. `#pragma omp parallel for reduction(min:min_c) reduction(max:max_c)`, or manual reduction with critical section, or similar).
- 5p: Fully parallel histogram computation (e.g with local histogram computation and parallel reduction of local histograms, or with a user declared reduction operator on the histogram vector, or similar).

Total:15p

Question 2: Dimensionality reduction with PCA (50 points)

Principal Component Analysis (PCA) is a classical method to perform dimensionality reduction and uncover structures in data. The method learns an orthogonal transformation that eliminates linear correlations and tries to capture as much variance in the data as possible. You are provided with a two dimensional toy dataset with $N = 1024$ samples plotted in Figure 1. The principal components can be computed using the covariance method, by constructing the covariance matrix of the data $C \in \mathbb{R}^{D\times D}$ and identifying its eigenvalue decomposition. The covariance matrix is given by

$$C = \frac{1}{N-1} X^T X \tag{3}$$

where $X \in \mathbb{R}^{N\times D}$ is constructed by stacking the dataset. Assuming that the input data are independent (but not uncorrelated), the covariance matrix is symmetric. The eigenvalue decomposition of the symmetric matrix C reads

$$C = V\Lambda V^{-1}, \tag{4}$$

where $\Lambda = \mathrm{diag}(\lambda_i)$ is a diagonal matrix with the eigenvalues stored in **descending** order. Due to the symmetry of the real covariance matrix, the eigenvectors are orthogonal to each other and they form an orthonormal basis, $V^{-1} = V^T$. The PCA components are the columns of the eigenvector matrix $V = [\mathbf{v_1}, \dots, \mathbf{v_D}]$. The transformed data read $\mathbf{y} = V^T\mathbf{x}$ and are linearly uncorrelated.

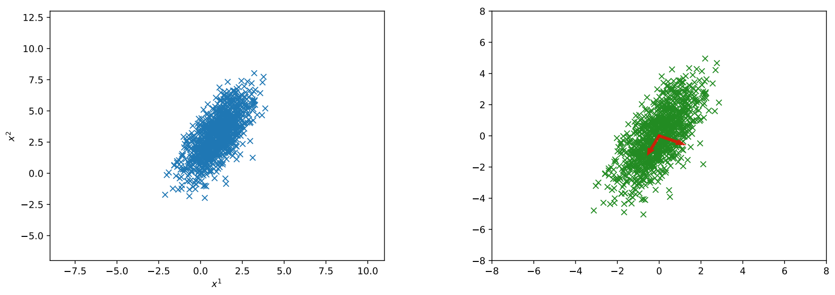

Figure 1: Toy two dimensional dataset along with its PCA components.

a) Use an appropriate routine provided by the LAPACK software library to compute the PCA of the dataset based on the covariance method. The Intel Math Kernel Library (MKL) includes a high-performance implementation of LAPACK. In order to use the MKL on Euler, you have to load the module with `module load mkl`. After loading the module, you can include the header `#include <mkl_lapack.h>` to access the LAPACK routines.

   Complete the steps in the skeleton code provided in the file `main_pca.cpp`. PCA is sensitive to the relative scaling of the input. For this exercise, **we consider relative input scaling to be relevant**, so the data should **not** be standardized (scaled with zero mean and unit variance). However, the data need to be centered. Write a program that performs the following tasks:

   1. read the provided dataset (saved in a memory allocation of $N \times D$, where N is the data-set size and D the data dimension)
   2. center the data

3. compute the covariance matrix of the data
4. call the `dsyev_()` routine of LAPACK to compute the eigenvalues of the matrix.

Can you recover the principal components plotted in Figure 1? Report the computed eigenvalues.

The implementation can be found in `homeworks/hw03/code_q2/solution_code/main_pca.cpp`.

Grading scheme:

- 3p for correctly transposing the data.
- 8p for correct mean (4p) and standard deviation (4p).
- 2p for correct normalization.
- 10p for correct covariance matrix.
- -2p for filling the whole matrix instead of only the upper/lower triangular part (due to symmetry).
- 5p for correct computation of the eigenvalues and eigenvectors using the Lapack routine.
- 2p Extraction of the components .
- 2p for correct compression ratio.
- 3p for correct reconstruction.

Total: 35p

b) In the following, we apply the PCA routine to a scenario closer to the real world. You are provided with a dataset with $N = 1280$ images of faces. Each face consists of a grayscale image $\mathbf{I} \in \mathbb{R}^{H \times W}$ that is flattened to a single data point $\mathbf{x} \in \mathbb{R}^D$, with $D = HW$, ignoring spatial structure. For this exercise $H = 50, W = 37$ so $D = 1850$. Perform PCA on this dataset and save **only** the $M = 10$ principal components (corresponding to the highest eigenvalues) in a `.txt` file. You have to save a matrix $V_r \in \mathbb{R}^{D \times M}$, obtained from the first M columns of the eigenvectors matrix $V \in \mathbb{R}^{D \times D}$ computed by PCA. Use the provided `python` routine to plot the principal components you computed.

Grading scheme:

- 5p for correct plot of the eigenvalues and/or eigenvectors (eigenfaces). In the case of the eigenfaces, note that they might be reversed in color, as the direction of the eigenvectors might be opposite. The $M = 10$ principal eigenfaces are shown in Figure 2.

Total: 5p

c) The components computed by PCA can be used to compress the dataset. The transformation to the compressed form reads $Z = XV_r$, where $Z \in \mathbb{R}^{N \times M}$. Use the computed $M = 10$ components to compress the data and **report** the compression ratio by measuring the number of floating point numbers needed to represent the original dataset and the compressed one (hint: for the purpose of compression you need to take into account the cost of the scaler, e.g. standardization).

The implementation can be found in
`homeworks/hw03/code_q2/solution_code/main_pca.cpp`.

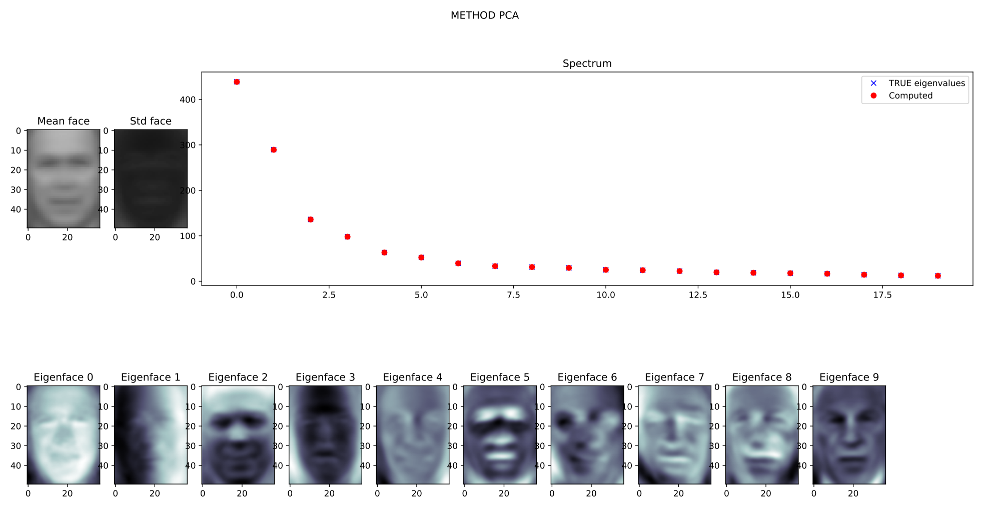

Figure 2: The $M = 10$ principal components (eigenvectors of the covariance matrix) corresponding to the highest eigenvalues.

Grading scheme:

- 5p for correct compression ratio. For $M = 10$ this is ≈ 67.65.

Total: 5p

d) Try to **reconstruct** the original dataset using the compressed data, the $M = 10$ PCA components and the data scaling factors and save the result in a `.txt` file. The reconstruction can be obtained by $X = ZV_r^T$. A `python` routine is provided that can be used to plot the components computed by your method, plot the reconstruction and benchmark (qualitatively) against a reference `python` implementation of PCA. **Validate** your result with the provided `python` routines.

Due to the small number of components included, the reconstruction only roughly resembles the original dataset. If we include more components, the reconstruction gets more accurate, at the cost of a lower compression ratio. For $M = 200$, the compression ratio is ≈ 3.76 (recall that for $M = 10$ the compression was ≈ 67.65). On the limit of a large data-set, only a few hundreds eigenfaces (eigenvectors) corresponding to the largest eigenvalues of the data covariance matrix would suffice to reconstruct the data in adequate accuracy.

Grading scheme:

- 5p for correct reconstruction, similarly to the one given in Figure 3.

Total: 5p

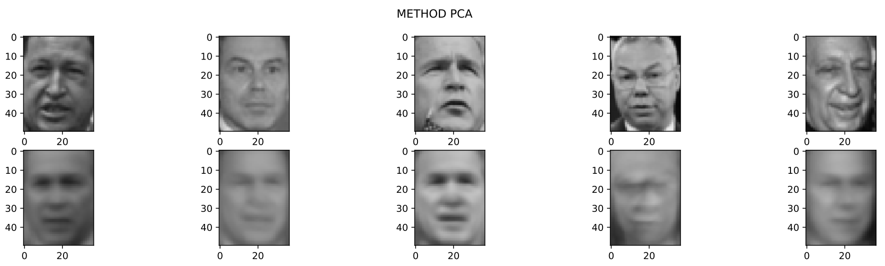

Figure 3: The reconstruction using **only** $M = 10$ principal components.

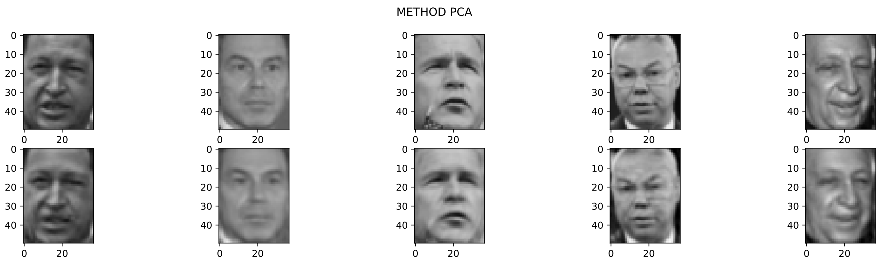

Figure 4: **(OPTIONAL)**The reconstruction using $M = 200$ principal components.