ETH zürich

High Performance Computing for Science and Engineering I

P. Koumoutsakos, W. Petersen, P. Hadjidoukas
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2017

Set 6 - PCA

Issued: November 03, 2017
Hand in (optional): November 10, 2017 8:00am

Question 1: Eigenvalues of Symmetric Matrices

In mathematics, particularly matrix theory, the $N \times N$ Lehmer matrix is the constant symmetric matrix defined by $A(i,j) = \min(i,j)/\max(i,j)$, where $i,j = 1..N$. For this exercise you will compute the eigenvalues of the Lehmer matrix, using an appropriate routine provided by the LAPACK and ScaLAPACK software libraries. The Intel Math Kernel Library (MKL) includes high-performance implementations of both libraries. 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.

$$A_2 = \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix};$$

$$A_3 = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1 & 2/3 \\ 1/3 & 2/3 & 1 \end{pmatrix};$$

$$A_4 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1 & 2/3 & 1/2 \\ 1/3 & 2/3 & 1 & 3/4 \\ 1/4 & 1/2 & 3/4 & 1 \end{pmatrix};$$

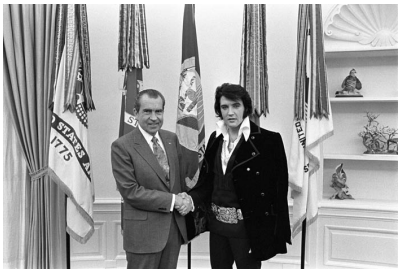
Figure 1: Examples of Lehmer matrices

Write a program that allocates and initializes an $N \times N$ Lehmer matrix and then calls the `dsyev_()` routine of LAPACK to compute the eigenvalues of the matrix. Measure the performance of `dsyev_()` using the multithreaded version of LAPACK for $N = 4096$ and 8192 and plot the speedup over the sequential version for all cases. Report the 5 largest computed eigenvalues for each case too. What do you observe? You can start with the skeleton code `lehmer.cpp`.

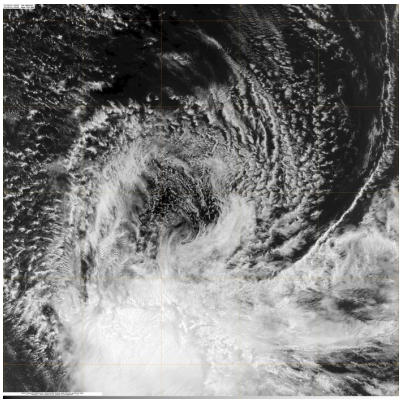
Question 2: Principal Component Analysis

Principal Component Analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

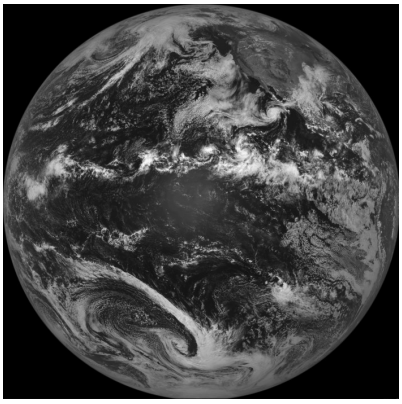
For this exercise you will implement PCA and then use it for compressing a gray-scale image. An image is represented as a 2-D matrix of double-precision numbers and stored row-wise as a compressed binary file. You do not need to uncompress the files; your code can directly read the compressed data by means of the zlib library, as in the following example:



(a) The King meets with president Nixon (469×700)



(b) Tropical cyclone image from NASA (4096×4096)



(c) Our planet (9500×9500)

```
#include <zlib.h> /* link with -lz */

// grows, fcols: dimensions of the stored image
// rows, cols: number of rows and columns that we will use
double *read_gzfile(char *fname, int frows, int fcols, int rows, int cols)
{
double *A;        // matrix
double *buf;    // temporary buffer
gzFile *fp;       // file descriptor

A = (double *)malloc(rows*cols*sizeof(double));
buf  = (double *)malloc(fcols*sizeof(double));

fp = gzopen(fname, "rb");
for (int i = 0; i < rows; i++) {
gzread(fp, buf, fcols*sizeof(double));
memcpy(&A[i*cols], buf, cols*sizeof(double));
}
gzclose(fp);

free(buf);
return A;
}
```

We provide three such binary files with varying size:

- `elvis.bin.gz`: the well-known Elvis-Nixon image (469×700 pixels).
- `cyclone.bin.gz`: image of a tropical cyclone (4096×4096 pixels), downloaded from the NASA Earth Observatory (`http://www.nasa.gov/mission_pages/hurricanes/archives/2011/h2011_Benilde.html`).
- `earth.bin.gz`: image of planet earth (9500×9500 pixels), downloaded from the NASA Earth Observatory (`https://earthobservatory.nasa.gov/IOTD/view.php?id=79383`).

The binary files for the corresponding images can be found in the `data` directory in the course Git repository (as well as on the course webpage). In addition, you will find a Matlab script file (`pca_demo.m`) that uses the Elvis-Nixon image and demonstrates the implementation steps that you have to follow for your C/C++ program. The code in the script is not optimized and its main purpose is to provide guidelines and help you in the verification of your results.

The first stage of PCA includes the construction of the covariance matrix. Your program initially reads the image matrix from the disk. Then, it computes the mean and standard deviation of the

features (image columns) and appropriately fills the covariance matrix C. Next, your program determines the eigenvalues and eigenvectors of the covariance matrix C. For this purpose you will use the `dsyev_()` LAPACK routine. Finally, your program computes the first K principal components.

a) Implement the PCA algorithm as described above. Apply PCA to the Elvis & Nixon test image and use the first $K = 1, 30, 50, 100$ principal components to get a series of compressed representations of the image. Compute and report the achieved compression ratio (your compressed data should include all the necessary information for a successful reconstruction). Start with the skeleton code in `pca.cpp`.

   In addition to the course material, you might find useful the provided Matlab script and the "Data Compression" section of the following webpage:
   `http://matlabdatamining.blogspot.ch/2010/02/putting-pca-to-work.html`.

b) Add multithreading capabilities to your PCA implementation and measure its performance for the all the test images. For this question you do not have to reconstruct and save the final image. Report the execution time of your program without including the time spent for reading the input image. How does your PCA implementation scale?