AM 213A, Winter 2022
Final Project (100 points)

Posted on Fri, Mar. 4, 2022
Hard deadline: 11:59 pm, Thu, Mar. 16, 2022

Submit your homework to your Git/Canvas by 11:59 pm, 3/16/2022

---

1. General Guidelines

1.1. Two parts

You have two parts in this homework set:

- **Part 1**: Numerical coding problems (make sure to use double precision)
  - **(a)**: Singular value problems for digital image compression (40 points)
  - **(b)**: Iterative methods for $\mathbf{Ax} = \mathbf{b}$ (60 points; 30 points each)
  - **Submission**: all source codes and reports to Git; a Git hash tag to Canvas
- **Part 2**: Theory problems – extra credits (15% towards the final grade)
  - **Submission**: all reports to *both* Git and Canvas; a Git hash tag to Canvas

Your final set of answers should consist of

- For Part 1: a pdf file with your written answers. The pdf must be created using LaTeX. Any relevant findings, coding issues (if any), and most importantly, summaries of your numerical test results, should be discussed here. No screenshots are allowed.
- For Part 2: solutions to the theory problems in a pdf file (a clean handwritten scanned pdf is allowed but a LaTeX document is preferred.).
- For both Part 1 and Part 2: Submit all your answers to your git. Use subdirectories named as **Part1/prob_a**, **Part1/prob_b**, and **Part2** under which you have **prob_a**, ..., **prob_e**. They are all under your top directory called **Project**.
- 15-page max for **Part 1**; 15-page max for **Part 2**.

Please organize and present the material in the best possible way, in particular, for your Part 1 solutions:

- For any written material, be informative but concise. Mathematical derivations, if appropriate, should contain enough pertinent steps to show the reader how you proceeded. All included figures should be clearly annotated and have an informative caption. Each problem should be addressed in its own Section (e.g., "Section 1: Problem 1", "Section 2: Problem 2", etc.). References should be included if you are using any material as part of your discussions and/or calculations.

- For the codes: All codes must be written in Fortran 90 (or more recent Fortran) or C, but *not* in any other language. MATLAB or Python is to be used to (i) generate figures, and/or (ii) perform simple analysis (e.g., calculating errors) the data produced by your Fortran/C codes. Please put all the codes relevant for each problem in separate directories called Prob1, Prob2, etc, as needed. Annotate your codes carefully so the reader knows what each part of the code does. For each routine and function in your Fortran/C module, include a header explaining the inputs and outputs of the code, and what the routine does (i.e., a comment section with explanations). In the main program in each directory, include a header that contains (i) the command required to compile the code, and (ii) the structure of the inputs and outputs of the main program.

Part 1: Mandatory Coding Problems

In your report, include inputs and outputs of your codes if needed. Describe sufficiently about the behaviors of your codes in each problem. Clarify what parts of your code implementations are corresponding to each individual problem.

IMPORTANT: Any code submissions *without* a Makefile will not be graded.

(a) SVD for image compression

Download the black-and-white image file `dog_bw_data.dat` and implement Fortran routines to compress the given image to lower resolutions using SVD. You will need to implement separate plotting routine(s) to visualize both the original and compressed images using either MATLAB or Python.

The data stored in `dog_bw_data.dat` is in double precision ascii format, with the values ranging from 0 (maps to black) to 255 (maps to white). The main goal is to conduct the following sequence of tasks:

1. Read-in `dog_bw_data.dat` as input from your Fortran/C routine. Call `dgesvd` routine in the LAPACK linear algebra library to perform a series of singular value approximations $\mathbf{A}_{\sigma_k}$ of the original data $\mathbf{A}$ stored in `dog_bw_data.dat`, i.e., the data in `dog_bw_data.dat` is treated as an $m \times n$ matrix $\mathbf{A}$. The pixel size, measured as width $\times$ height, of the original image is 5295×3355, which should be transposed in linear algebra to conform the conventional row $\times$ column dimension that gives 3355×5295 (i.e., $m = 3355, n = 5295$). It is though confusing in actual implementation depending on the column-major (e.g., Fortran) vs. row-major (e.g., C) formats. At this point, students should read `ref_8.pdf` under "Other Resources" for more details. Mathematically though, the rank of $\mathbf{A}$ is at most 3355.

2. Given $\mathbf{A}$, call `dgesvd` to compute the singular value decomposition of $\mathbf{A}$,

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are the left and right singular matrices, respectively. The full singular value matrix Σ is given as

$$\Sigma = \begin{pmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & & & \\ & & \ddots & & & & \\ & & & \ddots & & & \\ & & & & \sigma_r & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{pmatrix}. \tag{2}$$

To do this, you will first need to install LAPACK on your machine. The detailed instruction is available in the AM 129/209 online lecture note from F19. Go to the section "External Libraries for Scientific Computing" to learn how to install the library.

For those of you who have been using C, take a look at the past email communication (see the document `ref_1.pdf` under "Other Resources" on the Final Project page on Canvas) on useful guidelines and tips provided by TA Ian May last year.

There are other useful resources as well, including some plotting issues (`ref_4.txt`), the different conventions on how to refer to image sizes in the digital community vs. array sizes in linear algebra (read `ref_8.pdf`), and how to run Lapack on WSL (`ref_3.pdf`) and Hummingbird (`ref_7.pdf`). These are all very useful and please read them.

To successfully implement a driver routine that calls `dgesvd`, it is necessary to learn the structure of the routine. In particular, pay careful attention to the data types of the input and output arguments. You are welcome to use any resources (online or offline) to study them. One reference is:

- Ref. 1

Note: some may encounter an error message, saying *"Warning: Procedure 'dgesvd' called with an implicit interface at (1) [-Wimplicit-interface].* It is optional to remove this warning message, but in case this annoys you, you can include an interface block for `dgesvd` immediately after `implicit none` declaration. Make sure you first use appropriate debugging flags.

3. Reconstruct eight separate compressed (or low resolution) images (i.e., $\mathbf{A}_{\sigma_k}$, $k = 20, 40, 80, 160, 320, 640, 1280, 2560, 3355$). Each k-SVD approximation is defined by

$$\mathbf{A}_{\sigma_k} = \mathbf{U}\Sigma_{\sigma_k}\mathbf{V}^T, \tag{3}$$

where each $\mathbf{A}_{\sigma_k} \in \mathbb{R}^{m\times n}$ is a newly reconstructed compressed data from the original data $\mathbf{A}$. Each singular value matrix $\Sigma_{\sigma_k} \in \mathbb{R}^{m\times n}$ is a reduced matrix of the full diagonal matrix Σ containing only up to the first k largest singular values with $k \leq q = \text{rank}(\mathbf{A}) \leq 3355$, given as

$$\Sigma_{\sigma_k} = \begin{pmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & & & \\ & & \ddots & & & & \\ & & & \sigma_k & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{pmatrix}. \tag{4}$$

4. Save each of the eight data $\mathbf{A}_{\sigma_k}$ in an ascii format data file. Name your data file with reasonable file names such as `Image_appn_1xxxxx.dat` where the file index "`1xxxxx`" reflects the number of singular value used in each calculation, e.g., `100640` for $\Sigma_{\sigma_{640}}$.

5. Implement a plotting routine using either MATLAB or Python to visualize the compressed data. A good example of computing a similar image compression and displaying results in MATLAB is available here. Note that you should convert your double-precision data to an unsigned 8-bit data type (e.g., `uint8` in the MATLAB example) to properly display the results. Displaying data in double-precision in plotting will result in overflow errors.

6. Compute the corresponding errors (using MATLAB or Python) of each case using the averaged Frobenius norm, $||\cdot||_F$,

$$E_k = \frac{||\mathbf{A} - \mathbf{A}_{\sigma_k}||_F}{mn}, \tag{5}$$

   and plot the results as a function of the number of singular values.

In your PDF document:

- Report the first 10 largest singular values from σ_1 to σ_{10}. Also report the rest singular values of σ_k for $k = 20, 40, 80, 160, 320, 640, 1280, 2560, 3355$.

- Display all eight compressed images along with the original image. Identify figures clearly with the number of singular values used for each.

- Discuss what you see as you increase/decrease the number of singular values in calculations. Report at which k value you obtain an error lower than 10^{-3}.

(b) Iterative methods

(b.1) Gauss-Jacobi and Gauss-Seidel

In your `LinAl.f90` module, create

- a routine that solves the equation $\mathbf{Ax} = \mathbf{b}$ for a given $m \times m$ matrix $\mathbf{A}$, and an $m-$long vector $\mathbf{b}$ using the Gauss-Jacobi algorithm.
- a routine that solves the equation $\mathbf{Ax} = \mathbf{b}$ for a given $m \times m$ matrix $\mathbf{A}$, and an $m-$long vector $\mathbf{b}$ using the Gauss-Seidel algorithm.

Both routines should take as argument the required accuracy, and return the solution $\mathbf{x}$ once $||\mathbf{b} - \mathbf{Ax}||_2$ is smaller than the required accuracy. You are otherwise free to select the argument list, but please document it thoroughly. Within each routine, you need to print to a file the error $||\mathbf{b} - \mathbf{Ax}||_2$ at each iteration. Then, create a program that

- generates a matrix $\mathbf{A}$ full of ones, except on the diagonal where $a_{ii} = D$ where D is a value entered by the user (either as argument to the program, or as a prompt to the user),
- generates a vector $\mathbf{b}$ such that $b_i = i$,
- asks the user whether to use Gauss-Jacobi or Gauss-Seidel, and finally
- calls the right routine to solve $\mathbf{Ax} = \mathbf{b}$, with accuracy 10^{-5}, and prints the answer to the screen.

In your PDF document:

- Explain briefly what the Gauss-Jacobi and Gauss-Seidel algorithms do, how they differ, and what the convergence criterion for the algorithm is.
- Run the code for a 10×10 matrix $\mathbf{A}$ with $D = 2$, $D = 5$, $D = 10$, $D = 100$ and $D = 1000$. For each value of D where the algorithm converges, produce a figure showing on a log-linear plot, $||\mathbf{b} - \mathbf{Ax}||_2$ as a function of the iteration number (i.e. by plotting the file that you saved) for both Gauss-Jacobi and Gauss-Seidel (on the same plot).
- Discuss why some of the cases didn't converge, and compare the convergence rates of the two algorithms in light of the theoretical predictions discussed in the lectures.
- Finally modify the program so that $\mathbf{A}$ is the matrix $\mathbf{A}$ full of ones, except on the diagonal where $a_{ii} = i$. Run the program for a 10×10 matrix $\mathbf{A}$. Do either of the algorithms converge?

(b.2) Conjugate Gradient algorithm

In your `LinAl.f90` module, create the smart conjugate gradient algorithm without pre-conditioning to solve the problem $\mathbf{Ax} = \mathbf{b}$. The argument list should contain

- The input matrix $\mathbf{A}$ and input vector $\mathbf{b}$, as well as their dimension m (inputs).
- The desired accuracy (input).
- The solution (output).

The routine should check that the input matrix is symmetric. Modify your program from **(b.1)** as follow:

- Go back to using D as the diagonal elements.
- Add an option to use the Conjugate Gradient as the solver.

In your PDF document:

- Explain briefly what the Conjugate Gradient algorithms does and why it is guaranteed to converge in a finite number of iterations.
- Prove that the smart conjugate gradient algorithm is equivalent to the basic conjugate gradient algorithm (see notes).
- Run the code using the CG algorithm for a 10×10 matrix $\mathbf{A}$ with $D = 2$, $D = 5$, $D = 10$, $D = 100$ and $D = 1000$. What happens this time? Make a table comparing the number of iterations until complete convergence between the 3 algorithms, for each value of D. Explain the trends that you see.
- Explain why the diagonal pre-conditioner is not very useful for these matrices.
- Finally modify the program so that $\mathbf{A}$ is the matrix $\mathbf{A}$ full of ones, except on the diagonal where $a_{ii} = i$. Run the program for a 10×10 matrix $\mathbf{A}$, and a 100×100 matrix. In each case, does the CG algorithm converge? If yes, in how many iterations?
- (Extra credit): Implement the diagonal pre-conditioning, and re-do the last question. How much faster is the convergence this time?

Part 2: Optional Theory Problems – All theory problems are extra credits.

Problem 1. Consider the Gauss-Jacobi and Gauss-Seidel methods to solve $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{A} \in \mathbb{R}^{(p+q)\times(p+q)}$, defined by

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_p & -\mathbf{B} \\ -\mathbf{B}^T & \mathbf{I}_q \end{bmatrix} \tag{6}$$

where $\mathbf{I}_p$ and $\mathbf{I}_q$ are identity matrices of size $p \times p$ and $q \times q$ with $p \geq q$. Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_q > 0$ be the singular values of $\mathbf{B} \in \mathbb{R}^{p\times q}$.

(a) Define the iteration matrix $\mathbf{T}$ and the constant vector $\mathbf{c}$ for each Gauss-Jacobi when writing the two methods in the standard iterative form, $\mathbf{x}^{(k+1)} = \mathbf{T}\mathbf{x}^{(k)} + \mathbf{c}$.

(b) Repeat Part (a) for Gauss-Seidel.

(c) Show that the Gauss-Jacobi iteration matrix $\mathbf{T}$ has eigenvalues

$$\{\pm\sigma_i : 1 \leq i \leq q\} \cup \{0\}, \tag{7}$$

where the second set of zero means that the algebraic multiplicity of 0 is $p - q$, i.e., the characteristic equation is given as

$$p(\lambda) = \lambda^{p-q}(\lambda \pm \sigma_1)(\lambda \pm \sigma_2)\ldots(\lambda \pm \sigma_q). \tag{8}$$

(Hint: $\det \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \det(\mathbf{A}) \det(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})$ for any block matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$ with nonsingular $\mathbf{A}$.)

(d) Show that the Gauss-Seidel iteration matrix has eigenvalues

$$\{\sigma_i^2 : 1 \leq i \leq q\} \cup \{0\}, \tag{9}$$

where the algebraic multiplicity of 0 is p.

(e) Under what condition are those methods convergent? Between the Jacobi and Gauss-Seidel methods, which matrix would produce a faster method? Explain your answer.

Problem 2. Let $\mathbf{A} \in \mathbb{R}^{m\times m}$ be non-defective (so that there exist eigenvectors $\mathbf{v}_i$ corresponding to λ_i that form a basis) and let $\{\lambda_i\}_{i=1}^m$ be its eigenvalues. Assume $\lambda_1 = \cdots = \lambda_r$ for some $1 < r < m$ and

$$|\lambda_r| > |\lambda_{r+1}| \geq \cdots \geq |\lambda_m|. \tag{10}$$

Consider the Power Iteration method given by

$$\mathbf{x}^{(k+1)} = \frac{\mathbf{A}\mathbf{x}^{(k)}}{||\mathbf{A}\mathbf{x}^{(k)}||}, \tag{11}$$

with any arbitrary norm $||\cdot||$. Show that, for almost all possible choices of an initial vector $\mathbf{x}^{(0)}$, $\mathbf{x}^{(k+1)}$ will converge to an eigenvector associated to λ_1. For which values of $\mathbf{x}^{(0)}$ does the method *not* converge to such an eigenvector?

Problem 3. Let $\mathbf{A} \in \mathbb{R}^{m\times m}$ be symmetric and positive definite. Suppose that you have found the following algorithm that is known to produce a sequence of $\mathbf{A}_i$. The algorithm *claims* that $\mathbf{A}_i$ converges to a diagonal matrix $\Lambda = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_m)$, where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m > 0$ as $i \to \infty$. In each iteration, the algorithm uses the Cholesky decomposition to factorize $\mathbf{A}_i$ to an upper triangular matrix $\mathbf{U}_i$ whose diagonal elements are nonzero. The algorithm proceeds as follows:

Algorithm:
$\mathbf{A}_0 = \mathbf{A}$
`for` $i = 1, \cdots$
    $\mathbf{U}_i^T\mathbf{U}_i = \mathbf{A}_{i-1}$ **[Cholesky Decomposition]**
    $\mathbf{A}_i = \mathbf{U}_i\mathbf{U}_i^T$
`endfor`

Prove that the iteration is well-defined by showing the following steps:

(a) Show that $\mathbf{A}_i$ is also symmetric and positive definite to justify the use of Cholesky.

(b) Show that $\mathbf{A}_i$ is similar to $\mathbf{A}$ (i.e., $\mathbf{A}_i = \mathbf{B}^{-1}\mathbf{A}\mathbf{B}$ for some non-singular matrix $\mathbf{B}$).

(c) Justify why this iteration method is a valid eigenvalue revealing algorithm for $\mathbf{A}$.

Problem 4. Consider the Householder matrix defined by

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}. \tag{12}$$

(a) Show that for any nonzero vector $\mathbf{v}$, the matrix is orthogonal and symmetric.

(b) Let $\mathbf{a}$ be any nonzero vector and let $\mathbf{v} = \mathbf{a} + \alpha\mathbf{e}_1$, where $\alpha = \text{sign}(a_{11})||\mathbf{a}||_2$. Show that $\mathbf{H}\mathbf{a} = -\alpha\mathbf{e}_1$ by direct calculation.

(c) Determine $\mathbf{v}$ and α that transforms

$$\mathbf{H}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{13}$$

(d) Given the vector $\mathbf{a} = (2, 3, 4)^T$, specify a Householder transformation that annihilates the third component of $\mathbf{a}$.

(f) What are the eigenvalues of $\mathbf{H}$ for any nonzero vector $\mathbf{x}$?