Computational Linear Algebra 2022/23: First Coursework

Prof. Colin Cotter, Department of Mathematics, Imperial College London

This coursework is the first of three courseworks (plus a mastery component for MSc/MRes/4th year MSci students). Your submission, which must arrive before the deadline specified on Blackboard and Github Classroom, will consist of two components for the submission.

1. A pdf submitted on Blackboard to the Coursework 1 dropbox, containing written answers to the questions in this coursework.
2. Your code committed and pushed to your Github Classroom repository on the master branch. You can and should push at any time, but we will filter out commits made after the submission deadline (note that the time of the push is not relevant but please remember to push).

If you have any questions about this please ask them in the Ed Discussion Forum.

In answering the project work here, you will need to write additional code. This code should be added to your git repository in the cw1 directory. It is expected that it will just be run from that directory, so no need to deal with making it accessible through the installed module.

Some dos and don'ts for the coursework:

- If you have any personal issues that are affecting your ability to work on this course please **do** raise them with the course lecturer by email as soon as possible.
- **do** type the report and upload it as a machine-readable pdf (i.e. the text can be copied and pasted). This can be done by LaTeX or by exporting a PDF from Microsoft Word (if you really must). This is necessary to enable automated plagiarism checks.
- **do** describe in your answers in the report where to find the relevant code in your repository.
- **do** make regular commits and pushes, so that you have a good record of your changes.
- **do** remember to "git add" any new files that you add.
- **do** document functions using docstrings including function arguments.
- **do** add tests for your code, executable using pytest, to help you verify that your code is a correct implementation of the maths.
- **do** write your report as clearly and succinctly as possible. You do not need to write it as a formal report with introduction/conclusions, just address the questions and tasks in this document.
- **do** label and caption all of your figures and tables, and refer to them from the text by label (e.g. Figure 23) rather than relying on their position within the text (don't e.g. write "in the figure below). This is a good habit as this is a standard requirement for scientific writing.
- **don't** attach a declaration: it is assumed that it is all of your own work unless indicated otherwise, and you commit to this by enrolling on our degree programmes.
- **don't** post-process the pdf (e.g. by merging pdfs together) as this causes problems for the automated checks, and makes the resulting documents very large.
- **don't** write anything in the document about the weekly exercises, we will just be checking the code.
- **don't** include code in the report (we will access it from your repository).
- **don't** submit Jupyter notebooks as code submissions. Instead, **do** submit your code as .py modules and scripts.

- **don't** forget to git push your final commit!
- **don't** use "git add ." or add files that are reproducible from running your code (such as stored matrices, or .pyc files, etc.)
- **don't** use screenshots of code output. Instead, paste and format it as text.
- **don't** hide your answers to the questions in the code. The code is just there to show how you got your answers. Write everything in the report, assuming that the marker will only run your code to check that things are working.

Please be aware that both components of the coursework may be checked for plagiarism.

Coursework questions The coursework is organised so that it should be possible to reach a grade of 80% in the first two questions, and the third question is designed to be much more challenging. It is not expected that all candidates will attempt the third question.

The entire coursework can be done using real-valued floating point arithmetic (i.e. no complex numbers).

1. (25% of the marks)
   We will consider the set of basis functions $\{\phi_i(x)\}_{i=0}^{N-1}$, where
   $$\phi_i(x) = e^{-(x-x_i)^2/\Delta x^2}, \tag{1}$$
   with $x_i = i\Delta x$, and $\Delta x = 1/(N-1)$.
   (a) (**15%**) Add a Python function to the `cw1` directory that takes in a set of points $\{\hat{x}_i\}_{i=1}^{M}$, and observed function values $\{f_i = f(\hat{x}_i)\}_{i=1}^{M}$ for some unknown function $f(x)$. The Python function should return the basis coefficients obtained from a least squares fit to the basis (1). The Python function should have the option to use modified Gram-Schmidt or Householder for the QR factorisation, using your code from `cla_utils` in both cases.
   Add test(s) automatable through `pytest` that verify that the Python function works.
   (b) (**10%**) Investigate the behaviour of the fitted functions when $M = 100$ with either $N = 10$ or $N = 50$, applied to the function
   $$f(x) = \begin{cases} 1 & |x - 0.5| < 0.25, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$
   Make your investigations for equispaced points in the range [0,1], and for randomly generated points in the same range. In all cases investigate using modified Gram-Schmidt and Householder. Describe your results and present selected figures that illustrate them in your report.
2. (55% of the marks)
   This question concerns an array that is available at
   `https://raw.githubusercontent.com/comp-lin-alg/cw-data/main/A1.dat`
   Save this data to a file called `A1.dat`. You can read this file into a numpy array by using
   `A1 = loadtxt('A1.dat')`
   The result should be a 5×5 matrix stored as a numpy array.
   (a) (**9**%) Use your Householder QR factorisation code to compute the QR factorisation. How can we determine the rank of the matrix A_1 from the QR factorisation, and what is it?
   (b) (**9**%) It would be useful to automate finding the rank of a matrix efficiently (rather than spotting these features in the QR factorisation). One approach is modify the QR factorisation by column swapping, as follows. At the start of the kth iteration of the Householder algorithm, we swap the kth column with column k', with $k' \geq k$. The column k' is chosen so that the vector x in the Householder transformation has maximal norm. This means that we have to compute the norms of all the candidate x vectors and select the maximal one. After this swap, the Householder transformation proceeds as normal.
   Show that this procedure produces a matrix of the form
   $$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}, \tag{3}$$
   where R_{11} is an invertible upper triangular matrix $r \times r$ matrix, and r is the rank of A. (Hint: use a proof by induction on the Householder iteration number.)

(c) (**9**%) Implement this algorithm as code by adding an optional argument to `cla_utils.exercises3.householder`, by updating the interface to
`def householder(A, kmax=None, swap=None)`
If `swap` is `True` then `householder` should implement this extra step.[1] Add automatable tests.

(d) (**9**%) We add a further optional argument to `householder`, as follows. When the argument `reduced_tol` is present, `householder` should return R' given by

$$R' = \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}. \tag{4}$$

The argument `reduced_tol` should be a positive floating point number s. During the swap procedure, x vectors with norm $< s$ are considered to be zero. Add a function to your cw1 directory that uses this function to find the rank of a matrix (it should have a tolerance argument to pass to `reduced_tol`). Add a test automatable through `py.test` in the cw1 directory that verifies that this code is correctly implemented, by checking that it returns the correct rank for A_1 (don't assume that someone cloning your repository will have A_1 available, you need to provide it).

(e) (**9**%) Make some brief further investigations of the rank calculation using your own example matrices. One easy way to generate low rank matrices is as follows:

i. Randomly generate a matrix A.
ii. Compute the QR factorisation of A.
iii. Replace rows of R with zero (but not just the last ones, because this won't require swaps), to get R'. (You could also scale other rows of R to alter the size of that component.)
iv. Compute $B = QR'$.

You should investigate larger matrices than are more rank deficient, and investigate the impacts of the tolerance. Describe representative examples in your report, and add the script that produces them in `cw1`.

(f) (**10**%) A disadvantage of the swapping procedure is that we have to compute $n - k$ norms of $n - k$-dimensional arrays at each iteration k, which is quite a bit of computation (it requires to square $(n-k)^2$ numbers). Find a relationship between the norms of the arrays in iteration k and iteration $k+1$. Use this to propose a modification of the algorithm that only requires to square $(n-k)$ numbers. Implement your modification to `householder` when `swap` is true. Check that all of your tests still work.

3. (20% of the marks)

(a) (**4**%) For some problems it is useful to find a factorisation $RQ = A$ instead of QR. The following algorithm does this (in the case of square matrices).

- Reverse the rows of A (i.e. the first row becomes the last, the second row becomes the second last row, etc.), to get $\hat{A}$.
- Compute the QR decomposition of $\hat{A}^T$, to get $\hat{Q}$ and $\hat{R}$.
- Transpose $\hat{Q}$ and then reverse the rows, to get Q.
- Reverse the rows of $\hat{R}$, then transpose it, then reverse the rows again, to get R.

Show that this algorithm results in $A = RQ$ where Q is orthogonal and R is upper triangular.

(b) (**4**%) Implement this algorithm as a Python function in the `cw1` directory, providing automatable tests to verify that it works.

(c) (**3**%) Let A be an $n \times m$ matrix with $m \geq n$, and let B be a $n \times p$ matrix with $n \geq p$. We consider the simultaneous factorisation,

$$Q^T A U = R, \quad Q^T B = S, \tag{5}$$

where Q and U are $n \times n$ and $m \times m$ orthogonal matrices respectively, R takes the form

$$R = \begin{pmatrix} 0 & R_{11} \end{pmatrix}, \tag{6}$$

[1] In later parts of this question, there will be further modifications to this code. You should not need to preserve different versions for each of these, just update the original version each time. Just make sure any tests still work after further modifications.

where R_{11} is an $n \times n$ upper triangular matrix, and S takes the form

$$S = \begin{pmatrix} S_{11} \\ 0 \end{pmatrix}, \tag{7}$$

where S_{11} is a $p \times p$ upper triangular matrix.
Show that this factorisation can be computed using the QR and RQ factorisations of various matrices.

(d) (**3**%) Add code to `cw1` that implements this factorisation using your proposed procedure and provide automated tests.

(e) (**3**%) Now let A be an $m \times n$ matrix and B be a $p \times n$ matrix with $m \geq n \geq p$. Under the assumption that B has rank p and the null spaces of A and B only intersect at 0, show that this simultaneous factorisation can be used to solve the problem

$$\min_x \|Ax - b\|^2 \text{ such that } Bx = d. \tag{8}$$

(f) (**3**%) Add code to `cw1` that implements your proposed procedure for solving Equation (8). Provided automated tests.