Computational Linear Algebra 2021/22: Second Coursework

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

This coursework is the second 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, will consist of two components for the submission.

1. A pdf submitted to the Coursework 2 dropbox, containing written answers to the questions in this coursework.
2. A SHA tagging the revision of your repository, i.e. the repository for your weekly exercises that you have attempted so far. The SHA is the (nearly) unique hexadecimal code tagging each git revision. It is possible to find the SHA of a commit locally, in the copy of the repository on your computer. However this is a dangerous practice, because you might not have pushed that commit to GitHub, so you risk sending someone on a wild goose chase for a commit that they will never find. It is therefore a much better idea to grab the commit SHA for the commit you want directly from the GitHub web interface. See the link below for information on how to do this.
   `https://imperial-fons-computing.github.io/git.html#reporting-the-commit-hash`

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

The coursework marks will be assigned according to:

- 35% for the code for the weekly exercises from Section 3.1 to the end of Section 4.5. Only the exercises where you are asked to complete skeleton functions will be marked.
- 65% for the code and written answers to the project work here.

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:

- **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.
- **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.
- **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).
- **do** tell us which git commit is the one you want us to mark.
- **do** make regular commits and pushes, so that you have a good record of your changes.
- **don't** submit Jupyter notebooks as code submissions. Instead, **do** submit your code as .py modules and scripts.
- **do** remember to "git add" any new files that you add.
- **do** go onto Github and check that you haven't pushed any non-code files such as the content of your venv or the .pyc files that are automatically generated when running Python 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.
- **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** 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.
- If you have any personal difficulties affecting your work on this course please **do** raise them with the course lecturer by email as soon as possible.

Please be aware that both components of the coursework may be checked for plagiarism. It is fine to work together on the exercises and discuss your answers to project questions but you should write your own code and text, answering the project questions yourself.

Coursework questions

1. (35% of the marks)
   Complete the weekly exercises from Section 3.1 until the end of Section 4.5 and make sure that the code is committed your git repository and pushed to Github Classroom. Only the exercises where you are asked to complete skeleton functions will be marked.

2. (16% of the marks)

   (a) (5%) Design an algorithm with an $\mathcal{O}(n^2)$ operation count, for computing the $n \times n$ matrix $C = (xy^T)^k$ where $x, y \in \mathbb{R}^n$. Here the superscript k indicates a matrix power, e.g. $A^3 = AAA$. Show that your algorithm has an $\mathcal{O}(n^2)$ operation count.

   (b) (5%) We want to compute $D = A^T BA$ where $A \in \mathbb{R}^{m \times n}$, $B = \mathbb{R}^{m \times m}$. Compare the operation count of the algorithm using the formula $D = A^T(BA)$ with the operation count of the algorithm using the formula $D = (A^T B)A$. For which values of n and m does the second algorithm have a smaller operation count than the first algorithm?

   (c) Take 4 real $m \times m$ matrices, P, Q, R and S. Show how to compute the real and imaginary parts of $A = (P + iQ)(R + iS)$ using a total of three $m \times m$ matrix multiplications. Compute the operation count of this method (not just the leading order $\mathcal{O}$ term). (Hint: consider $T = (P + Q)(R - S)$.)

3. (17% of the marks) Here we consider the high school algorithm for finding the eigenvalues of a 2×2 matrix A:

   - Find the coefficients of the characteristic polynomial of A,
   - Find its roots using the classical quadratic formula.

   (a) (4%) What would it mean for this algorithm specifically to be stable?

   (b) (4%) What would it mean for this algorithm specifically to be backward stable?

(c) (4%) Write a Python script to apply this algorithm to the matrix

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1+10^{-14} & 0 \\ 0 & 1 \end{pmatrix}, \tag{1}$$

Compare the results to the exact solution, quantifying the error.

(d) Describe the expected scaling of this error with machine epsilon using material from the course.

4. (16% of the marks) A common matrix sparsity structure (the pattern of non-zero entries in the matrix) in higher-order numerical solution of PDEs is a structure of overlapping blocks. In this question we study a prototype version of this by considering $(4n+1) \times (4n+1)$ matrices of the type $A = I + \epsilon \sum_{k=1}^{n} B^k$, where

$$B^k_{ij} = \begin{cases} \nu_{ijk} & \text{if } 4(k-1) \le i,j \le 4k+1, \\ 0 & \text{otherwise.} \end{cases}, \quad k = 1, \ldots, n, \tag{2}$$

where ν_{ijk} are independent uniform random numbers and $0 < \epsilon < 0.1$ is a perturbation parameter.

(a) (3%) Apply unpivoted LU factorisation to matrices of this type, using your code from the exercises. Examine the sparsity structure of the resulting L and U matrices. How does it compare with the expected sparsity structure of banded matrices with the upper and lower bandwidths of A?

(b) (3%) Describe a modification of the banded matrix algorithm that exploits this additional structure, avoiding adding or multiplying by zero. Analyse the operation count for this approach.

(c) (3%) Implement this algorithm as code, providing appropriate automated tests.

(d) (3%) There is another way to exploit this special structure when solving $Ax = b$. Show how this system can be reduced to a system with a tridiagonal matrix by eliminating vector components. Describe an algorithm for solving $Ax = b$ by elimination, solving the reduced system and reconstructing the eliminated entries. Analyse the operation count (using $\mathcal{O}$) for this approach.

(e) (4%) Implement and demonstrate this second algorithm as code, including providing tests, preferably automatable using pytest.

5. (16% of the marks) In this question we consider a finite difference method for solving the stationary advection-reaction-diffusion equation,

$$\boldsymbol{b} \cdot \nabla u - \mu \nabla^2 u + cu = S(x,y), \tag{3}$$

where $\boldsymbol{b}$ is the (known) transport velocity, u is the (unknown) concentration of a chemical, μ is the diffusivity coefficient, c is the reaction coefficient, and $S(x,y)$ is a field describing the source of the chemical. The goal is to solve the equation for u, in this case in the domain $0 \le x, y \le 1$ with boundary conditions $u = 0$ when $x = 0$, $x = 1$, $y = 0$ and $y = 1$.
In the finite difference approximation, we consider a grid of points $(x_{i,j}, y_{i,j}) = (i\Delta x, j\Delta x)$ for $i = 0, \ldots, n$, $j = 0, \ldots, n$ and $\Delta x = 1/n$, for some positive integer n. Then, for each $0 < i, j < n$, we have values $\boldsymbol{b}_{i,j} = (b^1_{i,j}, b^2_{i,j}) = \boldsymbol{b}(x_{i,j}, y_{i,j})$ and $S_{i,j} = S(x_{i,j}, y_{i,j})$, and we seek $u_{i,j} \approx u(x_{i,j}, y_{i,j})$ such that

$$\frac{b^1_{i,j}(u_{i+1,j} - u_{i-1,j}) + b^2_{i,j}(u_{i,j+1} - u_{i,j-1})}{2\Delta x} - \mu \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{\Delta x^2} + cu_{i,j} = S_{i,j}, \tag{4}$$

for $0 < i, j < n$, with fixed values $u_{0,j} = u_{n,j} = u_{i,0} = u_{i,n} = 0$ corresponding to the boundary conditions.

(a) (3%) Consider the vector $\boldsymbol{v} \in \mathbb{R}^{(n-1)^2}$ with

$$v_{(n-1)(i-1)+j} = u_{i,j}, \quad 0 < i, j < n. \tag{5}$$

Formulate (4) as a matrix-vector equation for $\boldsymbol{v}$. What are the upper and lower bandwidths of the associated matrix A? What is the expected operation count for a banded LU factorisation algorithm to solve this problem, in the limit of large n.

(b) (3%) Implement the banded LU factorisation algorithm given in the course notes (this should be a minor modification of your unpivoted LU factorisation code) and demonstrate your expected operation count.

(c) (3%) The scaling of the operation count for banded LU factorisation in this problem with n is a problem for large n. Hence, it is tempting to consider iterative methods instead. In one approach, we construct iterative solutions $u^0_{i,j}$, $u^{1/2}_{i,j}$, $u^1_{i,j}$, $u^{1+1/2}_{i,j}$, $u^2_{i,j}$, ... starting from an initial guess $u^0_{i,j}$, according to the following iterative updates,

$$\begin{aligned} &\frac{b^1_{i,j}(u^{k+1/2}_{i+1,j} - u^{k+1/2}_{i-1,j})}{2\Delta x} \\ -\mu\frac{u^{k+1/2}_{i+1,j} + u^{k+1/2}_{i-1,j} - 4u^{k+1/2}_{i,j}}{\Delta x^2} + cu^{k+1/2}_{i,j} &= S_{i,j} - \frac{b^2_{i,j}(u^k_{i,j+1} - u^k_{i,j-1})}{2\Delta x} + \mu\frac{u^k_{i,j+1} + u^k_{i,j-1}}{\Delta x^2}, \end{aligned} \tag{6}$$

$$\begin{aligned} &\frac{b^2_{i,j}(u^{k+1}_{i,j+1} - u^{k+1}_{i,j-1})}{2\Delta x} \\ -\mu\frac{u^{k+1}_{i,j+1} + u^{k+1}_{i,j-1} - 4u^{k+1}_{i,j}}{\Delta x^2} + cu^{k+1}_{i,j} &= S_{i,j} - \frac{b^1_{i,j}(u^{k+1/2}_{i+1,j} - u^{k+1/2}_{i-1,j})}{2\Delta x} + \mu\frac{u^{k+1/2}_{i+1,j} + u^{k+1/2}_{i-1,j}}{\Delta x^2}. \end{aligned} \tag{7}$$

Describe a solution procedure for computing the $u^{k+1/2}_{i,j}$ values from the $u^k_{i,j}$ values, and then the $u^{k+1}_{i,j}$ values from the $u^{k+1/2}_{i,j}$ values according to these updates. In an algorithm making use of banded matrix LU factorisations, what is the operation count for one iteration from k to $k+1$?

(d) (3%) Implement such an algorithm using banded matrix LU factorisations that carries out this iterative scheme, applied to the problem where

$$\boldsymbol{b} = \alpha(-\sin(\pi x)\cos(\pi y), \cos(\pi x)\sin(\pi y)), \quad S = s_0 \exp\left(-(x-1/4)^2/r_0^2 - (y-1/4)^2/r_0^2\right), \tag{8}$$

for parameters α, s_0, r_0. The iteration should terminate when the solution satisfies the equation up to a chosen tolerance. Provide tests to verify that the code is a correct implementation, for a choice of parameters where the method converges well.

(e) (4%) Investigate how quickly the iterative scheme reaches an accurate solution of (4) when different values of n, α, s_0, r_0, c, are chosen, and describe your conclusions of this investigation. Produce some example images of how the solution looks with various parameters.