CS319: Scientific Computing (with MATLAB)

# PROJECTS (Updated 04-Apr-2023)

Niall Madden

Week 9, Semester 2, 2023

https://www.niallmadden.ie/2223-CS319/2223-CS319-Projects.pdf

# 1: Overview

Your project will consist of

1. An idea you select/propose, ideally in one-to-one discussion with Niall in labs or lectures.
2. An outline for what you plan to achieve. This will be presented as 250 word project proposal. (Word count is indicative: I won't check).
3. Your own implementation of some algorithm, coded in MATLAB, and with your own choice of test problem(s) so you can verify properties of the algorithm, such as its complexity, or convergence, etc.
4. A report, written as a MATLAB live script that explains the problem, describes the algorithm, and draws conclusions about its properties.

You can choose to work alone, or with one other person (assuming they agree!).

# 2: Project topics

1. At the end of this presentation, I'll outline a selection of topics you might consider working on. However, I also encourage you to propose your own ideas.
2. Discuss your idea with Niall, ideally in person today! Failing that, discuss by email (though you will still have to discuss the proposal in person). Once agreed, enter
3. **Deadline**: submit your agreed project topic by **5pm, Tuesday 14 March**, at the latest. Do this on Blackboard ... 2223-CS319...Projects... Project Topic. If in a group, name the other person.

## Some ideas for projects

The topics listed here have the purpose of stimulating discussion. More will be added to this as I think of them. When choosing a topic, you are welcome to suggest one of your own, or pick one of these. However, you must discuss with me No two people will be allocated the same topic, unless working as a group. So, don't assume topics below are still available.

1. **Graphs:** The graph isomorphism problem.
2. **Graphs:** Testing planarity.
3. ~~**Graphs**: Computing the chromatic number of a graph, and visualising the result.~~ [MS]
4. **Graphs**: Visualising graphs (e.g., using the Graph Lapacian).
5. ~~**Linear algebra**: Computing the Choleskey factorisation of a matrix, and explain its applications in solving linear systems and/or determining positive definiteness.~~ [MC]

6. ~~**Linear algebra**: Your own implemenation of Conjugate Gradients (and/or GMRES), and examples of how and when it works (and fails).~~ [AB]
7. **Linear algebra**: Schur compliments for solving linear systems.
8. ~~**Linear algebra**: Estimation of eigenvalues and eigenvectors, e.g., combining The Power Method and the Rayleigh Quotient.~~ [DH]
9. ~~**Linear algebra**: Computing and visualising Gerschgorin Disks. (See Nick Higham's blog)~~ [EL]
10. **OOP**: Your own implementation of arbitrary precision integers, with applications, e.g., in cryptography or primality testing.
11. **OOP**: Computing with polynomials.
12. ~~**Image processing**: compression~~ [EC]
13. ~~**Image processing**: Computing and applying the SVD.~~ [CP]
14. ~~**Image processing**: Denoising via linear codes.~~ [RP+MT]
15. **Data analysis**: $k$-means or $k$-nearest neighbours clustering
16. ~~**Optimization:** Newton's and Secant methods in 2D~~ [CDMcL]

17. **Optimization:** Linear programming.
18. **Interpolation:** Adaptive construction of interpolation points for high-order polynomial interpolation.
19. **Interpolation:** Adaptive construction of interpolation points for piecewise polynomial interpolation.
20. ~~**Interpolation:** Interpolation and sampling of functions in 2D.~~ [SJ]
21. ~~**Differential equations:** Solution of high-order ODEs by high-order or implicit Runge-Kutta methods.~~ [AN+RC]
22. ~~**Differential equations:** Time-stepping methods for PDEs~~ [MC]
23. **Differential equations:** Solution of boundary value problems by finite difference or finite element methods.
24. ~~**Differential equations:** Simulating SIR models~~ [WJ]
25. ~~**Differential equations:** Finding and classifying equilibria~~ [OC]
26. **Quadrature:** Derivation and comparison of schemes, such as Gauss-Lobotto methods.
27. **Information theory:** Shannon Entropy.

28. ~~**Computational Physics:** Mechanics and bouncing balls~~ [JC]
29. **Other:** Exploration of, and exposition on, the Chebfun toolbox.
30. **Other:** Mixed precision computing.
31. ~~**Other:** Implementation and analysis of QuickSort in a OOP setting.~~ [RW]
32. ~~**Other:** Music generation using Markov chains.~~ [DL]
33. ~~**Other:** Loosless compression with FLAC~~ [JN]

More sources:

- **Algorithms from the Book**: https://epubs-siam-org.nuigalway.idm.oclc.org/doi/book/10.1137/1.9781611976175
- Projects in O'Leary's book (See Weeks 1 and 2).
- Chapter 7 of "Learning MATLAB".
- Chapter 26 of "The MATLAB Guide".
- Nick Higham's blog: https://nhigham.com/blog/

# 3: Project proposal

1. The project proposal outlines what you plan to achieve.
2. It should begin with a (tentative) title, your name, ID and email address, and the name of your collaborator (if any).
3. The proposal should describe what you plan to do, explain why it is interesting, and describe how you plan to test it.
4. It should include a set of milestones, and a time-line for each. (It more important that you make such as plan, than actually following the time-line!)
5. 250 words should suffice. (Word count is indicative: I won't check).
6. This will be uploaded to Blackboard as a PDF.
7. **Deadline: 5pm, Tuesday, 21 March**.
8. If your proposal is OK (and on time), you get full marks for that part. If not, you can resubmit until you do. (That is: we want to get this right).

# 4: The code

The code for your project is a chance for your to show your skills in MATLAB programming, and in scientific computing.

Examples of things you might try to demonstrate:

- skill in writing modular code. At the very least there should be one function, but I expect there will be multiple functions.
- is your code *vectorised*? Can any `for` loops be avoided?
- is your code *efficient*? Have you identified any bottlenecks, e.g., using the profiler?
- examples of writing your own class will be very welcome.
- can you use an external data source or produce one (and show skill in data input/output).
- Reusing unattributed code will be very unwelcome.

1. Code should be submitted on Blackboard (see the link in the `Project` section). If you prefer to share it via a git repository, please do. Make sure the repo is private and shared with me. In addition, add a link to it as text in the submission section of Blackboard.
2. If your code features multiple files, please submit them as a `zip` file, or similar archive. This is because MATLAB uses file-name as function/class names. However, Blackboard likes to rename all the files.
3. Only MATLAB files should be submitted. You can use standard `.m` files or Live Scripts, as you prefer. I think the best solution is to have functions and classes as `.m` files, and then a single `.mlx` file that shows how it works. That `.mlx` file might also contain the report, but that is not required. However, it is OK to write everything in single file, if you like.
4. Every **code** file you submit (other than 3rd party libraries) should, without exception, have comments at the start with
   - your name, ID and email address

- comments that briefly summarising what the code does and how to use it.

You are not required to add you name, etc, to data files that are part of the project.

5. Make sure all files have valid names: starts with a letter or underscore, and contains only letters, digits and underscopres. Nothing else is allowed. (If the name is not valid, not only will I not be able to run it as-is, it implies you did not try running the code yourself!).
6. You may use built-in functions in MATLAB, or third-party functions, so long as correct attribution is used. (For example, last year a group used a MATLAB package for automatically downloading data from Yahoo Finance). But your project must contain a substantial amount of code that you have written.

7. There is no lower or upper limit in the number of lines your code must have. My experience is that good projects often feature about 200 lines of code for algorithms, functions, classes, etc, and about 100 lines of code for demonstrating that it works.
8. Code that you claim is written by you should be written by you. Don't use anyone else's code without attribution. *If plagiarism is identified in a project, it will receive a score of zero*.

Broadly, your code will be assessed for the extent for which it shows your skills in programming in MATLAB.

In doing that, I will pay particular attention to the following:

- Is there competent use of basic programming structures, such as `for` loops and `if` statements?
- Is there appropriate use of data-types, with conversion between them as needed?
- Are matrices and vectors handled correctly, and efficiently, E.g., is the code vectorised, are sparse matrices used where appropriate?
- Have you shown you can work with external data sources, e.g., reading data from text files, or importing from images or spread-sheets, we scraped from the web?
- If relevant, good use live script features, such as controllers?
- Any used of advanced language features, such as writing classes, or an app?

- **Important:** demonstrated skills in scientific computing, including the implementation of algorithms, and correct use of tools.
- **Important:** You project should have at least one function, ideally several. Consider using advanced techniques in function programming, such as variable input and output argument lists, or specifying argument properties.

# 5: The report

Your report will be submitted as a PDF document, ideally (but not necessarily) generated from a MATLAB live script.

It should start with:

- project title
- your name, ID number, email address
- names of anyone you collaborated with.

Suggested organisation:

1. **Introduction:** A summary of what you have done. This should be 300-500 words, and written in a non-technical style: anyone should be able to understand it. The emphasis should be on the problem solved, and why it is interesting, and not on the code.

# 5: The report

2. **Code:** A technical discussion of the code. Outline the major algorithms you've implemented, functions you've written, or classes you've developed. This can be quite "high-level": you don't have to explain what individual lines of code do, but rather focus on the big picture: *what was the main challenge to be addressed?*
3. **Case use:** An example of typical input and output. Having read this, the use should know enough to test your code for their own examples.
4. **Discussion:** A discussion on what you have learned/discovered. What have you discovered about this algorithm and/or this method? **What steps have you taken to verify its properties?**. For example, if relevant, have you established the rate of convergence, or estimated the run-time for inputs of given sizes?
5. **Highlights:** What are the highlights of your project? What took the most effort? What are you most proud of?

# 5: The report

6. **Conclusion:** Details any limitations of your code that you have noted, and a comparison with your original project proposal. If you had more time to work on your project, what would you do next? List any references used.

**For group projects:**

- Each member of the group must submit copies of the code and report.
- The report must state the members of the group, and **what aspects of the project that each contributed to.**

Submit a PDF version of the report (which will be be checked with TurnItIn) and, separately, the live script (if there is one).

If you've already submitted the Live Script as part of your code, there is no need to re-submit it along with the report.

Your report will be assessed for the extent for which it explains your approach to problem solving, and conveys an understanding of the problem and your solution method.

*The list below gives a rough marking scheme: I won't follow it exactly, because different projects have different strengths. For example, some focus on very challenging problems, but use pretty standard methods. Some involve highly complex algorithms, but applied to generic problems. Most are a balance.*

1. **Introduction:** Does the introduction explain the problem you aimed to solve, and why one would be interested? This section is probably the most important: if someone was reading this out of interest (rather than having to grade it), are they likely to continue reading? [$\approx$ 4 Marks]
2. **Code:** Having read this, I should have a sense of the key-building blocks of your code. While reading this, I'll also be reviewing the code. [$\approx$ 3 Marks]

3. **Case use:** Having read this, I should be able to re-use your code for my own examples. [≈ 3 Marks]
4. **Discussion:** Depth of insights about the problem or solution, and properties of the algorithm/implementation. [≈ 3 Marks]
5. **Highlights:** [≈ 1 Marks]
6. **Outlook/Conclusion:** [≈ 1 Marks]

# 6: Time-line and grading

Your projects will contribute 40% to your over-all grade for CS319. The break-down of marks is as follows.

(a) Negotiating the project topic with Niall completed by 17:00, Tuesday 14 March [**5 Marks**]

(b) The project proposal submitted by 17:00, Tuesday 21 March. [**5 Marks**]

(c) Code [**15 Marks**]

(d) Project report [**15 Marks**]

Deadline for the code and report: **5pm Thu 6 April**.