About your project

Goal. The goal of the Math 661 project is for you to become more familiar with certain optimization problems and algorithms than is possible with the brief coverage that is typical of the rest of the course.

Expectations. Your project may be application-driven (*choose the problem(s) first*) or algorithm-driven (*choose the algorithm(s) first*)—see page 3—but all projects must include a specific optimization problem and a specific optimization algorithm. That is, you will implement at least one algorithm, i.e. in MATLAB/PYTHON/ JULIA/etc., and apply your code to at least one example problem.

Both numerical computation and mathematical analysis are required. For the latter you will analyze the problem(s) and algorithm(s) using theory from the textbook[1] and/or from other references. Once your code is running you should provide some empirical (numerical experimentation) evidence regarding the error and/or performance of your algorithm(s). Analysis is important because it shows you have absorbed ideas from the course, and because it distinquishes between algorithms. Numerical evidence is important because it confirms that you understood the algorithm well enough to implement it correctly.

Your problem(s) must be in the following form:

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{array}{ll} g_i(x) = 0, & i \in \mathcal{E}, \\ g_i(x) \ge 0, & i \in \mathcal{I}, \end{array} \tag{1}$$

though of course you may replace $\min$ with $\max$. If your project is algorithm-driven then you must identify which such problems are solved by your algorithms(s).

Form (1) describes a very large class of problems, but note your problem(s) must be finite-dimensional ($\mathbb{R}^n$), though it may arise from an infinite-dimensional source. The problem(s) must be well-enough understood to allow you to both precisely identify the objective function $f(x)$ and precisely identify a feasible set S defined by finitely-many constraints $g_i(x)$. It is o.k. if there are no constraints.

Due dates. There are two due dates for the project:

I = Proposal: **Part I is due Friday 9 November at the start of class.**[2] There are no format requirements for this part except that it **can be at most two pages**.

The proposal should precisely say what problem(s) or algorithm(s) you want to look at. If application-driven it should explain *briefly* where the optimization problem(s) came from, and in any case it should briefly motivate your proposed choice(s) of algorithm(s). Then the proposal should talk though what the complete project will contain, to the degree possible.

[1]Griva, Nash, and Sofer, *Linear and Nonlinear Optimization*, 2nd ed., SIAM Press 2009.

[2]This is a change from the date on the syllabus.

Several quality references are *expected*; online references are o.k.[3] Make specific references to our textbook when that is appropriate.

Spending at least a few hours on thinking and research at this stage can be very effective, but you should spend at most (say) 6 hours on this part. It would be good to use a version control system already at this stage, if you know how.

II = Project: **On Tuesday 11 December at 5pm you will submit the complete project.**

It should have the format and section headings as shown on page 4 below. This format corresponds to a LaTeX document template already posted online:

`bueler.github.io/M661F18/blankproject.tex`

Of course you do not have to use LaTeX, but please do use the indicated section headings.

The total length **must be 20 pages or less**; I will not accept longer projects. The total time spent on the whole project should be at most 25 hours.

Choosing a topic. One of my jobs will be to help you choose problem(s) and algorithm(s) so that your project has reasonable difficulty. I will likely advise you not to bite off too much! The bigger the scope the easier it is to get lost in the application, the algorithmic details, or in difficulties with codes or analysis. And your proposal allows me to give good feedback on the topic, perhaps a gentle nudge in the direction of a nice variation in topic, or a different analysis to consider, etc.

You may **not** choose a topic which has been, or will be, adequately covered in the course. For example, though the basic simplex method is not a good topic. However, implementations of the simplex method which respect sparsity, a topic we do not cover, would be a great choice (Chapter 7). Similarly, the classical Newton method is not a good subject but investigating quasi-Newton methods beyond BFGS, or line search methods beyond backtracking, or trust region methods, would all be good choices (Chapters 11,12).

Here are three approaches to choosing a topic if you don't already have one:

Approach 1: Inspiration from the Wikipedia page on mathematical optimization. See the "Applications," "Major subfields," and "Computational ... techniques" sections of

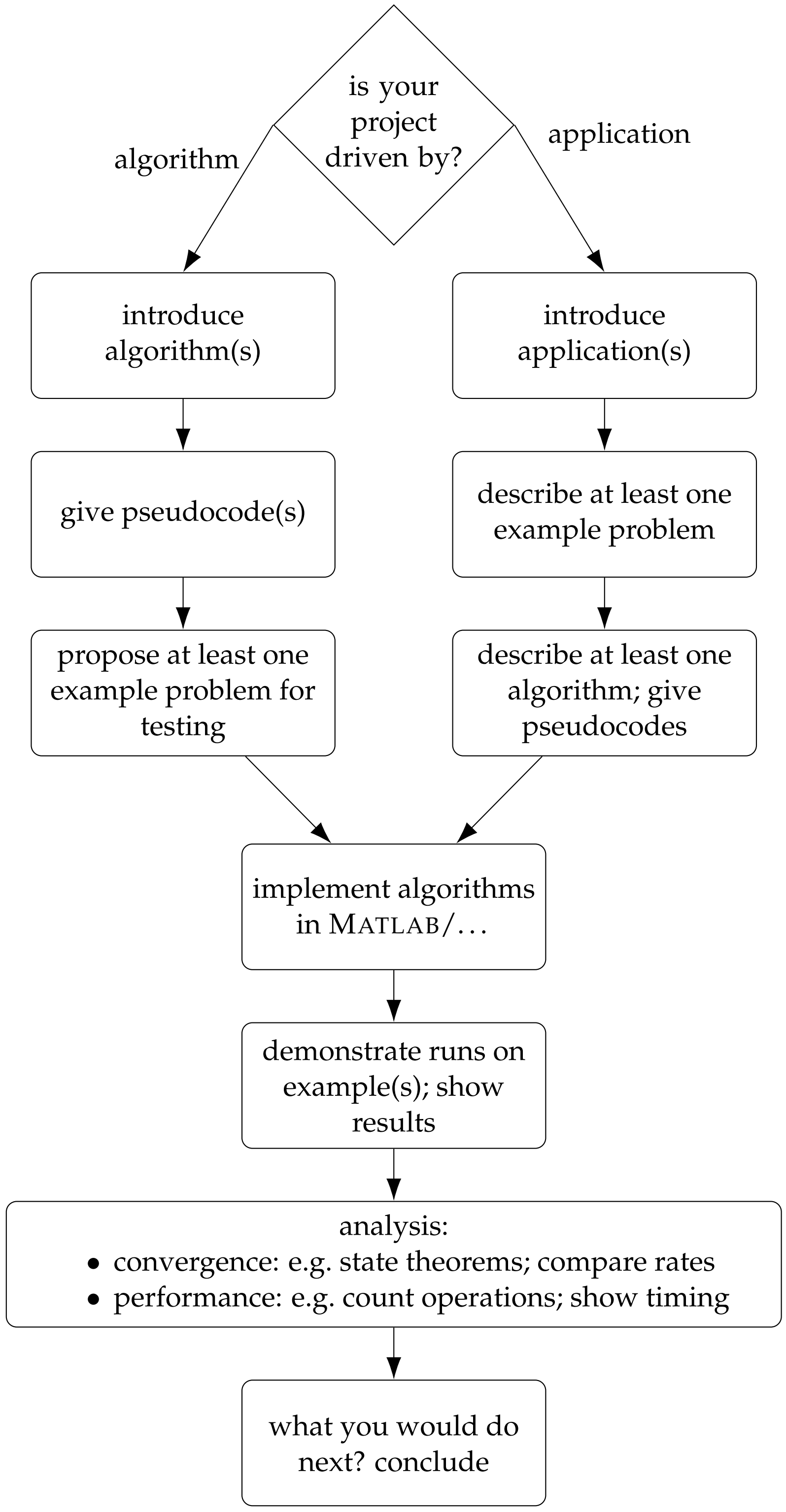
`en.wikipedia.org/wiki/Mathematical_optimization`

Approach 2: Investigate skipped material from the textbook. Consider section(s) that you find interesting and which we did not cover. However, don't just choose a section at random; the above Wikipedia page is better for starting from scratch.

Approach 3: A topic related to your thesis (if you have one). You can talk to me, but it will take a while for me to understand the context of your problem. It would be better to talk to your thesis advisor. It is reasonable to ask "are there optimization problems related to my expected thesis"? Broadly-speaking these might relate to optimal design or parameter fitting, or to algorithms which arise in your field of interest. There may be a paper to read about optimization in your field. **Don't** cover territory comparable to your whole thesis; extract a little part and do it carefully.

[3] However, I find that many informal online documents are of low quality.

Structure of the project. Here is a rough flow-chart. It aligns well with the section headings on the next page.


is your project driven by?
algorithm
application
introduce algorithm(s)
introduce application(s)
give pseudocode(s)
describe at least one example problem
propose at least one example problem for testing
describe at least one algorithm; give pseudocodes
implement algorithms in MATLAB/...
demonstrate runs on example(s); show results
analysis:
• convergence: e.g. state theorems; compare rates
• performance: e.g. count operations; show timing
what you would do next? conclude