CS 169/268 Introduction to Optimization

Syllabus, Fall 2017

Instructor: Eric Mjolsness

emj@uci.edu

Rough outline of topics (*= optional):

1. **Introduction to (introduction to) optimization**
    a. **1D unconstrained opt methods**
    b. **problem definitions**
        i. **zoo of opt problems**
            1. **discrete vs. continuous *(& interior point methods)***
            2. **local vs. global opt; local vs. global convergence**
            3. **unconstrained vs. convex vs. constrained**
        ii. **structure & topology of spaces (domain, range, function spaces)**
        iii. **special cases in opt:**
            1. **linearity, quadraticality, convexity, sampled objectives, …**
                a. **in objectives vs. in constraints**
            2. **integer- or discrete-valued solutions**
                a. **combinatorial optimization**
2. **Unconstrained optimization**
    a. **optimality conditions**
    b. **nonderivative methods**
        i. **Nelder-Mead, Simulated Annealing, Genetic Alg.s, Diff. Evol.*, …**
    c. **convergence rates & condition number**
    d. **gradient methods (conjugate gradients; block coordinate descent; …)**
    e. **Newton & quasi-Newton methods**
    f. **multiscale/multigrid methods ***
3. **Equality-constrained optimization**
    a. **optimality conditions: Lagrange multipliers**
    b. **gradient projection**
    c. **augmented Lagrangian method, & lasso ***
4. **Discrete optimization**
    a. **combinatorial optimization**
        i. **Linear programming: Simplex and Interior Point methods**
        ii. **branch and bound ***
        iii. **linear & quadratic assignment**
        iv. **computational complexity & NP-completeness ***
    b. **interior-point methods**
5. **Inequality-constrained optimization**

    a. **optimality conditions: Kuhn-Tucker**
    b. **barrier & penalty methods**
    c. **duality**
    d. **gradient methods eg. active sets ***
    e. **convex optimization ***
6. **Application areas ***
    a. **logistics & operations research**
    b. **mechanical & electrical engineering**
    c. **machine learning**
    d. **computer vision**
    e. **robotic planning, at multiple levels**
7. **Advanced topics (** probably not in class, but investigate for final project?)**
    a. **Nondifferentiable problems:**
        i. **subgradient methods**
        ii. **cutting planes**
    b. **Trust region methods**
    c. **Constraint-specific methods, eg. orthogonal matrices**
    d. **application class: Finite Element Method**
    e. **Algebraic multigrid**
    f. **scaling up**

Assignments and Grading:

For undergraduates : There will be a several homework sets and possibly quizzes worth 40%, a midterm exam worth 30% and a group project worth 30%.

For graduate students: There will be a several homework sets and possibly quizzes worth 40%, a midterm exam worth 30% and a group project worth 30%. The graduate student work will be more extensive by about x2, and more advanced.

Programming will be in Python, except possibly for the final project.

The Group Final Projects will be described in a separate document.

Midterm exam: Tuesday, November 2, in class.

See “Workload” below.

Schedule:

Instructor absences (some with substitutes): Oct. 26, Nov 23 (University Holiday), Dec. 5 likely.

Final Projects due: TBD, but likely poster presentations in class on Dec 7 and all electronic materials submitted by the regularly scheduled Final Exam time for this class. There will be a physical poster session (attendance is mandatory for all students, not just representatives of each group), and a written group report due together with source code and an electronic form of the poster. All group members must be named.

References

Everyone should get access to at least one good optimization book, somehow. Here (below) are some possibilities.

Strongly Recommended:

A. Belegundu & T. Chandrupatla, Optimization Concepts and Applications in Engineering, 2nd ed. Cambridge U. Press.

Optional:

D. Bertsekas, Nonlinear Programming, 2nd ed. Athena Scientific. (For more serious students of numerical optimization.)

R. Baldick, Applied Optimization, Cambridge U. Press. (More elementary and less complete, but contains many good examples.)

Alternatives and background reading on special topics:

S. Boyd, Convex Optimization. (Somewhat specialized to … convex optimization.)

K. Lange, Optimization, Springer 2nd ed. *(A statistics-oriented viewpoint on optimization. Useful treatments of analysis (Ch 2), EM methods (Ch 8-9), Lasso (Ch 13), and calculus of variations (ch 17).)*

C. Papadimitriou, Combinatorial Optimization. (Somewhat specialized to … combinatorial optimization; also a bit dated.)

D. Luenberger, Linear and Nonlinear Programming. (A bit dated, but classic.)

W. Press et al., Numerical Recipes, Cambridge U. Press. (***Warning***: Contains lots of actual, useful working code - and therefore must *not* be consulted on some but not all of our homeworks or homework problems!! But generally you get a chance later on to compare with good external code. All of it is available online.)

Deeper theory background:

D. Luenberger, Optimization by Vector Space Methods, Wiley Interscience Press. (What we are *really* doing in optimization is working in various function spaces.)

Software resources

Python will be language for all or nearly all homework problems. In early homeworks you will not be able to use external Python optimization codes, but instead you must write your won. Later on, scipy and sympy can be used from your code.

For group final projects, open or academically licensed software such as Mathematica, Matlab, R, … mathematical Problem-Solving Environments (PSEs) have built-in optimization capabilities. Gurobi is specific to optimization. Many open source codes also exist. But sometimes you will be asked to make your own implementation, from zero starting code; only afterwards is it fair to compare it with other implementations. Generally you may use small amounts of non-executable pseudocode *if* you fully cite them. Whether you can also use executable code (eg in PSEs) and/or read other people's source code depends on the individual assignment, and it must always be with full attribution.

("Code" is at least: Any expression in any formal language L which can be compiled into or interactively interpreted as a computer program by software specific to language L. Pseudocode looks like code and/or mathematical notation, but is either not formalized, or cannot be automatically compiled or interpreted by any language-specific software you have access to.)

TA/Reader

UCI did not allocate a TA or Reader for this year's offering of this class. Therefore, you may expect much reduced feedback concerning your work compared to previous years, and much higher reliance on your own personal efforts and responsibility. *If this is a problem for you, then please do not take this class.*

Mathematics

We will assume working knowledge of calculus, linear algebra (e.g. $A \cdot \mathbf{x} = \mathbf{b}$; $A \cdot \mathbf{x} = \lambda \mathbf{x}$), multivariable calculus (e.g. multivariable Taylor's theorem; $\partial_j x_i = \delta_{ij}$), and some lightly reviewed matrix theory(e.g. SVD = singular value decomposition of a matrix); also discrete mathematics, ability to read and write a proof in logic notation, subscript/index notation (e.g. $[A \cdot \mathbf{x}]_i = \sum_j A_{ij} x_j$; $\sum_j \delta_{ij} f(j) = f(i)$; $||\mathbf{x}||^2 = \sum_i (x_i)^2$ which is just the Pythagorean theorem in index notation), and ability to program in Python.

Many if not most of the present-day breakthroughs in machine learning, artificial intelligence, computer graphics, computer vision, scientific computing, etc. that seem to be transforming world culture actually rely on applications of mathematical numerical optimization (not to mention statistics and several other applied mathematical fields). Key to the very nature of numerical optimization is that it is quite mathematical. Correspondingly, …

From some more concrete points of view this class is highly mathematical, as is the Instructor. If this is a problem for you, then *please do not take this class.* If you do not like mathematical

abstraction, or if you can only really understand something through very concrete examples, then *please do not take this class.* In these cases, and in the absence of strong UCI TA/Reader support, this class is not for you. Of course the Instructor will provide motivating examples in lectures, but the examples themselves will become more abstract and more abstractly presented as the course moves along.

On the other side of the coin, if you are a budding mathematician then you may find yet more mathematical and less computational courses in other departments worth considering as an alternative.

Workload

As detailed above there will be 4-5 homeworks that involve mathematical programming, and a mathematics-centric midterm exam, and a final group project. Grading will generate little feedback and little capability for review, for reasons described above.