2018-06-12

1 Introduction

The title of this course is "Numerical Methods for Data Science." What does that mean? Before we dive into the course technical material, let's put things into context. I will not attempt to completely define either "numerical methods" or "data science," but will at least give some thoughts on each.

Numerical methods are algorithms that solve problems of continuous mathematics: finding solutions to systems of linear or nonlinear equations, minimizing or maximizing functions, computing approximations to functions, simulating how systems of differential equations evolve in time, and so forth. Numerical methods are used everywhere, and many mathematicians and scientists focus on designing these methods, analyzing their properties, adapting them to work well for specific types of problems, and implementing them to run fast on modern computers. Scientific computing, also called Computational Science and Engineering (CSE), is about applying numerical methods — as well as the algorithms and approaches of discrete mathematics — to solve "real world" problems from some application field. Though different researchers in scientific computing focus on different aspects, they share the interplay between the domain expertise and modeling, mathematical analysis, and efficient computation.

I have read many descriptions of *data science*, and have not been satisfied by any of them. The fashion now is to call oneself a data scientist and (if in a university) perhaps to start a master's program to train students to call themselves data scientists. There are books and web sites and conferences devoted to data science; SIAM even has a new journal on the Mathematics of Data Science[1]. But what is data science, really? Statisticians may claim that data science is a modern rebranding of statistics. Computer scientists may reply that it is all about machine learning[2] and scalable algorithms for large data sets. Experts from various scientific fields might claim the name of data science for work that combines statistics, novel algorithms, and new sources of large scale data like modern telescopes or DNA sequencers. And from my biased perspective, data science sounds a lot like scientific computing!

[1] And I'll be excited to read the first issues!

[2] The statisticians could retort that machine learning is itself a modern rebranding of statistics, with some justification.

Though I am uncertain how data science should be defined, I am certain that a foundation of numerical methods should be involved. Moreover, I am certain that advances in data science, broadly construed, will drive research in numerical method design in new and interesting directions. In this course, we will explore some of the fundamental numerical methods for optimization, numerical linear algebra, and function approximation, and see the role they play in different styles of data analysis problems that are currently in fashion. In particular, we will spend one week each talking about

- Optimization methods for ML.
- Latent factor models, factorizations, and analysis of matrix data.
- Function approximation and kernel methods.
- Numerical methods for graph data analysis.

You will not need to have a prior numerical analysis course for this course, but you should have a good grounding in calculus and linear algebra. I have posted some background notes to remind you of some things you may have forgotten, and perhaps to fill in some things you may not have seen. Please do ask questions as we go, and if you see anything that you think should be corrected or clarified, send me an email (or you can suggest a change on the course GitHub repository).

2 Optimization

Much of this class[3] will involve different types of optimization problems:

$$\text{minimize } \phi(x) \text{ s.t. } x \in \Omega. \tag{1}$$

Here $\phi : \mathbb{R}^n \to \mathbb{R}$ is the *objective function* and Ω is the *constraint set*, usually defined in terms of a collection of constraint equations and inequalities:

$$\Omega = \{x \in \mathbb{R}^n : c_i(x) = 0, i \in \mathcal{E} \text{ and } c_i(x) \leq 0, i \in \mathcal{I}\}.$$

A point in Ω is called *feasible*; points outside Ω are *infeasible*. In many cases, we will be able to solve *unconstrained* problems where Ω is the entire domain of the function (in this case, all of $\mathbb{R}^n$), so that every point is feasible.

[3]There are also some topics in the class that do not fit naturally into an optimization framework, and we will deal with them as they come.

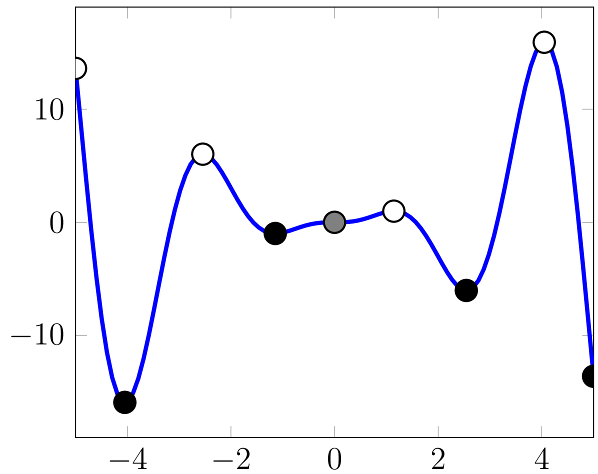

Figure 1: The objective $\phi(x) = x^2 \sin(2x)$ on $\Omega = [-5, 5]$ has four local minima (black), along with four maxima (white) and one critical point which is neither (gray). Most optimizers will only find one of the local minima, unless they are provided with a good initial guess at the global optimum.

Even simple optimization problems need not have a solution. For example, a function might not be bounded from below (e.g. the identity function $x \mapsto x$ on $\Omega = \mathbb{R}$), or there might be an asymptotic lower bound that can never be achieved (e.g. the function $x \mapsto 1/x$ on $\Omega = \{x \in \mathbb{R} : x > 0\}$). If ϕ is continuous and Ω is closed and bounded (i.e. a *compact* subset of $\mathbb{R}^n$), then at least there is some $x_* \in \Omega$ that solves the global optimization problem problem: that is, $\phi(x_*) \leq \phi(x)$ for all other $x \in \Omega$. But just because a solution exists does not mean it is easy to compute! If all we know is that ϕ is continuous and Ω is compact, any algorithm that provably converges to the global minimizer must eventually sample densely in Ω[4]. This statement of gloom is usually too pessimistic, because we generally know more properties than simple continuity of ϕ. Nonetheless, in many cases, it may be too expensive to solve the global optimization problem, or at least to prove that we have solved the problem. In these cases, the best we know how to do in practices is to find a good *local* minimizer, that is, a point $x_* \in \Omega$ such that $\phi(x_*) \leq \phi(x)$ for all $x \in \Omega$ close enough to x_*. If the inequality is strict, we call x_* a *strong* local minimizer.

[4]See *Global optimization* by Törn and Žilinskas.

The picture is rosier when we want to solve a *convex* problem; that is,

1. The *set* Ω is convex: $\forall x, y \in \Omega$, we have $\alpha x + (1-\alpha) y \in \Omega$ for $0 < \alpha < 1$.
2. The *function* ϕ is convex on Ω: for any $x, y \in \Omega$ and $0 < \alpha < 1$,
$$\phi\left(\alpha x + (1-\alpha)y\right) \leq \alpha\phi(x) + (1-\alpha)\phi(y).$$
If the inequality is strict, we say ϕ is *strongly* convex.

For a convex problem, every local minimizer is also a global minimizer, and the local minimizers (if there is more than one) form a convex set. If the function ϕ is strongly convex, then there is only one minimizer for the problem. Moreover, we have simple algorithms that we can prove converge to the solution of a strongly convex problem, though we might still decide we are unhappy about the cost of these methods for large problems.

Whether or not they are convex, many of the optimization problems that arise in machine learning and data science have special structure, and we can take advantage of this structure when we develop algorithms. For example:

- Among the simplest and most widely used optimization problems are *linear programs*, where
$$\phi(x) = c^T x$$
subject to constraints $Ax \leq b$ and $x \geq 0$. Among their many other uses, linear programs are a building block for *sparse recovery* methods in which we seek to represent a signal vector as a linear combination of a small number of elements in some dictionary set. We will not discuss sparse recovery in detail, but will touch on it when we discuss *matrix completion* next week.

- Unconstrained problems with *quadratic* objective functions
$$\phi(x) = \frac{1}{2} x^T A x + b^T x + c$$
are another simple and useful type. A common special case is the *linear least squares* objective
$$\phi(x) = \frac{1}{2}\|Ax - b\|^2 = \frac{1}{2} x^T A^T A x - b^T A x + \frac{1}{2} b^T b.$$

We constantly optimize quadratic functions, both because they are useful on their own and because optimization of quadratics is a standard building block for more complicated problems. Optimizing a quadratic objective is the same as solving a linear system, and so we can bring to bear many methods of modern linear algebra when solving this problem. For example, a particularly popular approach is the *conjugate gradient* method.

- In many cases, the objective is a sum of simple terms:
  $$\phi(x) = \sum_{i=1}^{n} \phi_i(x).$$
  An important case is the *nonlinear least squares* problem $\phi(x) = \|f(x)\|^2$, which we will discuss later this week. In modern machine learning, problems of this form are often solved by various *stochastic gradient* methods.

- Most *spectral* methods in data science can be phrased in terms of the *quadratically constrained quadratic program*
  $$\phi(x) = \frac{1}{2}x^T Ax + b^T x + c, \quad \Omega = \{x \in \mathbb{R}^n : x^T Mx = 1\}.$$
  We will see such problems in matrix data analysis and also graph clustering and partitioning methods. We can sometimes create methods for these problems that build on the fact that we have good methods for solving eigenvalue problems.

- Some nonconvex objectives are *bi-convex*: $\phi(x_1, x_2)$ is a convex function of x_1 for a fixed x_2 and vice-versa, though not in x as a whole. We will see these types of problems repeatedly when we consider analysis of matrix data. We can sometimes create methods for these problems based on the idea of *block coordinate descent* (also known as *nonlinear Gauss-Seidel* or *alternating iterations*) that solve a sequence of convex subproblems in each of the variables in turn.

- We also consider problems where ϕ (and possibly Ω) depend on an additional parameter s; for example, in an optimization problem coming from regression, we might have an additional *regularization parameter*. In this case, we might consider *continuation* methods that compute the curve of solutions.

3 Optimality conditions

In an unconstrained problem with a differentiable objective function, a necessary (but not sufficient) condition for x_* to be a local minimizer is that $\phi'(x_*) = 0$. For intuition, picture a function $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$; if you'd like to be concrete, let $n = 2$. Absent a computer, we might optimize ϕ by the physical experiment of dropping a tiny ball onto the surface and watching it roll downhill (in the steepest descent direction) until it reaches the minimum. The statement that $\phi'(x_*) = 0$ (or that $\nabla\phi(x_*) = 0$) basically means the function looks flat at x_* to a sufficiently near-sighted observer; if $\phi'(x_*)$ is not zero, then $x_* - \epsilon\nabla\phi(x_*)$ will be a little bit "downhill" of x_*; that is, if $\|\nabla\phi(x_*)\| \neq 0$ then

$$\phi(x_* - \epsilon\nabla\phi(x_*)) = \phi(x_*) - \epsilon\|\nabla\phi(x_*)\|^2 + o(\epsilon) < \phi(x_*)$$

for sufficiently small ϵ.

Most students learn the first-order optimality conditions for unconstrained optimization in a first course, but sometimes that course gets everyone too stuck on the idea of computing a gradient. What is really happening is that the function should be "flat in all directions," i.e. all directional derivatives are zero. This is equivalent to the statement that the gradient is zero, of course, but sometimes it is notationally easier to check that an arbitrary directional derivative is zero than to try to write down the gradient. For example, consider the quadratic objective

$$\phi(x) = \frac{1}{2}x^T Ax + x^T b + c.$$

Now, we will write an arbitrary directional derivative of ϕ in terms of "variational notation" (described in the background notes):

$$\delta\phi(x) = \left.\frac{d}{d\epsilon}\right|_{\epsilon=0} \phi(x + \epsilon\delta x) = (\delta x)^T (Ax + b).$$

At a critical point, $\delta\phi(x)$ should be zero for any choice of δx, so the stationary point occurs at $Ax_* + b = 0$. There is a unique minimizer x_* if A is positive definite. When the number of variables is not too large — up to a few thousand, say — we might solve this system of linear equations directly using a variant of Gaussian elimination if we wanted to find the minimizer. When the number of variables is much larger, we may prefer to use an iterative method to solve the system, e.g. the method of conjugate gradients

(CG). This method can be interpreted either as an iterative solver for linear equations or as an iterative optimization method.

Now let's turn to the *constrained* case. Rather than repeating the formal derivation of the first-order constrained optimality conditions that you have likely seen before, let me again give you an interpretation that involves some physical intuition. For the unconstrained case, we thought about solving the problem by rolling a tiny ball down hill until it came to rest. If we wanted to solve a constrained minimization problem, we could build a great wall between the feasible and the infeasible region. A ball rolling into the wall would still roll freely in directions tangent to the wall (or away from the wall) if those directions were downhill; at a constrained miminizer, the force pulling the ball downhill would be perfectly balanced against an opposing force pushing into the feasible region in the direction of the normal to the wall. If the feasible region is $\{x : c(x) \leq 0\}$, the normal direction pointing inward at a boundary point x_* s.t. $c(x_*) = 0$ is proportional to $-\nabla c(x_*)$. Hence, if x_* is a constrained minimum, we expect the sum of the "rolling downhill" force ($-\nabla \phi$) and something proportional to $-\nabla c(x_*)$ to be zero:

$$-\nabla \phi(x_*) - \mu \nabla c(x_*) = 0.$$

The *Lagrange multiplier* μ in this picture represents the magnitude of the restoring force from the wall balancing the tendency to roll downhill.

More abstractly, and more generally, suppose that we have a mix of equality and inequality constraints. We define the *augmented Lagrangian*

$$L(x, \lambda, \mu) = \phi(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \sum_{i \in \mathcal{I}} \mu_i c_i(x).$$

The *Karush-Kuhn-Tucker (KKT) conditions* for x_* to be a constrained minimizer are

$$\begin{aligned}
\nabla_x L(x_*) &= 0 & & \\
c_i(x_*) &= 0, \quad i \in \mathcal{E} & & \text{equality constraints} \\
c_i(x_*) &\leq 0, \quad i \in \mathcal{I} & & \text{inequality constraints} \\
\mu_i &\geq 0, \quad i \in \mathcal{I} & & \text{non-negativity of multipliers} \\
c_i(x_*)\mu_i &= 0, \quad i \in \mathcal{I} & & \text{complementary slackness}
\end{aligned}$$

where the (negative of) the "total force" at x_* is

$$\nabla_x L(x_*) = \nabla \phi(x_*) + \sum_{i \in \mathcal{E}} \lambda_i \nabla c_i(x_*) + \sum_{i \in \mathcal{I}} \mu_i \nabla c_i(x_*).$$

The complementary slackness condition corresponds to the idea that a multiplier should be nonzero only if the corresponding constraint is active (a "restoring force" is only present if our test ball is pushed into a wall).

Like the critical point equation in the unconstrained case, the KKT conditions define a set of (necessary but not sufficient) nonlinear algebraic equations that must be satisfied at a minimizer. I like to think about the "rolling downhill" intuition for these necessary conditions because it suggests a way of thinking about numerical methods.

For completeness, we will say a few brief words about the second-order *sufficient* conditions for optimality. In the unconstrained case, x_* is a strong local minimizer of ϕ if $\nabla \phi(x_*) = 0$ and the Hessian matrix H_ϕ is positive definite; that is because in this case x_* is the strong minimizer of the quadratic approximation

$$\phi(x) \approx \phi(x_*) + \frac{1}{2}(x - x_*)^T H_\phi(x_*)(x - x_*).$$

In the constrained case, the Hessian only needs to be positive definite for those u that are orthogonal to $\nabla c_i(x_*)$ for each c_i that is active (has a nonzero Lagrange multiplier). We will see this idea in two weeks when we talk about kernel methods, and in particular talk about the idea of a *conditionally positive definite* kernel function.

4 Numerical methods

With our lightning review of some fundamental theory out of the way, it is time to turn to numerical methods! In particular, we will spend the next three lectures talking about gradient and stochastic gradient methods, Newton and Gauss-Newton, and (block) coordinate descent. We will see additional solver ideas as we move through the semester, but these are nicely prototypical examples that illustrate two running themes in the design of numerical methods for optimization.

Fixed point iterations All our nonlinear solvers (and some of our linear solvers) will be *iterative*. We can write most as *fixed point iterations*

$$x^{k+1} = G(x^k), \tag{2}$$

which we hope will converge to a fixed point, i.e. $x^* = G(x^*)$. We often approach convergence analysis through the *error iteration* relating the error $e^k = x^k - x^*$ at successive steps:

$$e^{k+1} = G(x^* + e^k) - G(x^*). \tag{3}$$

As a teaser for this sort of analysis, consider one of the simplest algorithms I know: gradient descent with a fixed step size h, applied to the quadratic model problem

$$\phi(x) = \frac{1}{2} x^T A x + b^T x + c$$

where A is assumed to be symmetric and positive definite. The algorithm produces iterates

$$\begin{aligned} x^{k+1} &= x^k - h\nabla\phi(x^k) \\ &= x^k - h(Ax^k + b) \\ &= (I - hA)x^k - hb. \end{aligned}$$

Now we subtract the fixed point equation for the true solution x^* in order to get an error iteration:

$$\begin{array}{r} [x^{k+1} = (I - hA)x^k - hb] \\ -[x^* = (I - hA)x^* - hb] \\ \hline =[e^{k+1} = (I - hA)e^k] \end{array}$$

where $e^k = x^k - x^*$. The error iteration converges iff the largest eigenvalue of A is less than $2h^{-1}$; if this condition is satisfied, then

$$\|e^{k+1}\| \leq (1 - h\lambda_{\max}(A))\|e^k\|$$

and so we have $\|e^{k+1}\| \leq (1 - h\lambda_{\max}(A))^{k+1}\|e^0\|$, a convergence rate which is known as (R)-linear convergence or as geometric convergence, depending on which corner of the literature one prefers to read.

Model-based methods Most nonquadratic problems are too hard to solve directly. On the other hand, we can *model* hard nonquadratic problems by simpler (possibly linear) problems as a way of building iterative solvers. The most common tactic — but not the only one! — is to approximate the nonlinear function by a linear or quadratic function and apply all the things we know about linear algebra. We will return to this idea in the next lecture when we discuss Newton-type methods for optimization.