CS 6241: Numerics for Data Science

Introduction

David Bindel

2025-01-21

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. 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[1] 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

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

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!

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 roughly two weeks each talking about

- Linear algebra and optimization concepts for ML.
- Latent factor models, factorizations, and analysis of matrix data.
- Low-dimensional structure in function approximation.
- Function approximation and kernel methods.
- Numerical methods for graph data analysis.
- Methods for learning models of dynamics.

You will not strictly need to have a prior numerical analysis course for this course, though it will help (the same is true of prior ML coursework). But you should have a good grounding in calculus and linear algebra, as well as some "mathematical maturity". I have posted some to remind you of some things you may have forgotten, and perhaps to fill in some things you may not have seen. In addition, the readings section of the home page consists of a number of basic (and not-so-basic) texts to which you can refer. Along with course notes, we will be using chapters from some of these books (and sometimes research papers) as required reading. 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.

What does a matrix mean?

Linear algebra objects and matrix representations

I like to think about four fundamental objects in linear algebra involving maps on or between abstract vector spaces $\mathcal{V}$ and $\mathcal{U}$:

1. A *linear map* $\mathcal{A} : \mathcal{V} \to \mathcal{U}$ satisfies $\mathcal{A}(v + w) = \mathcal{A}v + \mathcal{A}w$ and $\mathcal{A}(\alpha v) = \alpha \mathcal{A}v$ for any vectors $v, w \in \mathcal{V}$ and scalar α.
2. An *operator* $\mathcal{A} : \mathcal{V} \to \mathcal{V}$ represents a mapping of a space onto itself.

3. A *bilinear form* $a : \mathcal{V} \times \mathcal{U} \to \mathbb{R}$ is linear in both the first and the second argument. If $\mathcal{V}$ and $\mathcal{U}$ are vector fields over $\mathbb{C}$, it is natural to instead consider *sesquilinear forms*, which are linear in the first argument and in the conjugate of the second argument.

4. A *quadratic form* $\phi : \mathcal{V} \to \mathbb{R}$ is $\phi(v) = a(v, v)$ where $a : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a bilinear form (real case) or sesquilinear form (complex case).

All four of these objects appear in various applications in data analysis. Linear maps between different spaces are a basic building block for regression and function approximation; operators are used to describe linear time invariant systems, such as Markov chains; bilinear forms model the similarity between pairs of objects represented by vectors v and u; and quadratic forms are used to measure a variety of quantities of interest in network analysis, such as cut sizes and edge densities.

The abstract objects of linear algebra can be realized concretely as matrices with a choice of bases. One way of thinking of a basis is as an invertible map from a concrete vector space (like $\mathbb{R}^n$) to an abstract vector space (like $\mathcal{V}$). Taking this perspective, we write a basis for $\mathcal{V}$ as the "quasimatrix"

$$V = \begin{bmatrix} v_1 & \ldots & v_n \end{bmatrix}$$

where each column is a vector in $\mathcal{V}$, allowing us to write an expansion of a general vector $v \in \mathcal{V}$ compactly as

$$v = \sum_{j=1}^{n} v_j x_j = Vx.$$

We can similarly write a basis for $\mathcal{U}$ as the quasimatrix

$$U = \begin{bmatrix} u_1 & \ldots & u_m \end{bmatrix}.$$

With this notation in place, we have the following matrix representations

1. We represent a linear map $\mathcal{A} : \mathcal{V} \to \mathcal{U}$ by the matrix $A = U^{-1}\mathcal{A}V$. Then the abstract operation $u = \mathcal{A}v$ is equivalent to the concrete matrix-vector product $y = Ax$, where $u = Uy$ and $v = Vx$.

2. We represent a bilinear form $a : \mathcal{V} \times \mathcal{U} \to \mathbb{R}$ as $a(Vx, Uy) = y^T Ax$.

3. We represent an operator $\mathcal{A}$ by $A = V^{-1}\mathcal{A}V$ — just as with a linear map between two spaces, but restricted to a single choice of basis.

4. We represent a quadratic form $\phi : \mathcal{V} \to \mathbb{R}$ as $\phi(Vx) = x^T Ax$ for some symmetric matrix A.

Canonical forms and decompositions

Given a good choice of basis, we can find matrix representations with very simple forms, sometimes called *canonical forms*. For general choices of matrices, the canonical forms are

- For a linear map, we have the canonical form

$$A = U^{-1}\mathcal{A}V = \begin{bmatrix} I_k & 0 \\ 0 & 0 \end{bmatrix}$$

  where k is the rank of the matrix and the zero blocks are sized so the dimensions make sense.

- For an operator, we have the *Jordan canonical form*,

$$J = V^{-1}\mathcal{A}V = \begin{bmatrix} J_{\lambda_1} & & \\ & J_{\lambda_2} & \\ & & \ddots\, J_{\lambda_r} \end{bmatrix}$$

  where each J_λ is a Jordan block with λ down the main diagonal and 1 on the first superdiagonal.

- For a quadratic form, we have the canonical form

$$a(Vx) = \sum_{i=1}^{k_+} x_i^2 - \sum_{i=k_++1}^{k_++k_-} x_i^2 = x^T A x, \quad A = \begin{bmatrix} I_{k_+} & & \\ & -I_{k_-} & \\ & & 0_{k_0} \end{bmatrix}.$$

  The integer triple (k_+, k_0, k_-) is sometimes called the *inertia* of the quadratic form (or *Sylvester's inertia*).

As beautiful as these canonical forms are, they are terrible for computation. In general, they need not even be continuous! However, if V and U have inner products, it makes sense to restrict our attention to orthonormal bases. This restriction gives canonical forms that we tend to prefer in practice:

- For a linear map, we have the canonical form

$$U^{-1}\mathcal{A}V = \begin{bmatrix} \Sigma_k & 0 \\ 0 & 0 \end{bmatrix}$$

  where k is the rank of the matrix and the zero blocks are sized so the dimensions make sense. The matrix Σ_k is a diagonal matrix of *singular values*

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_k > 0,$$

  and the bases U and V consist of the *singular vectors*.

- For an operator, we have the *Schur canonical form*,

$$V^{-1}\mathcal{A}V = T$$

  where T is an upper triangular matrix (in the complex case) or a quasi-upper triangular matrix that may have 2-by-2 blocks (in the case of a real matrix with complex eigenvalues). In this case, the basis vectors span nested invariant subspaces of $\mathcal{A}$.

- For a quadratic form, we have the canonical form

$$a(Vx) = \sum_{i=1}^{n} \lambda_i x_i^2 = x^T \Lambda x,$$

  where Λ is a diagonal matrix with $\lambda_1, \ldots, \lambda_n$ on the diagonal.

If we compute canonical forms for matrices (rather than for abstract operators), we have some of the standard matrix decompositions that appear in numerical linear algebra:

- The Singular Value Decomposition (SVD):

$$A = U\Sigma V^*$$

- The Jordan decomposition (square A):

$$A = VJV^{-1}$$

- The Schur decomposition (square A):

$$A = VTV^*$$

- The symmetric eigendecomposition (symmetric A)

$$A = V\Lambda V^*$$

When A is symmetric, the latter three decompositions are the same. When A is in addition positive semi-definite, all four decompositions coincide. In general, though, the "right" canonical decomposition depends on the type of linear algebraic object we are working with.

Optimization

Much of this class[2] will involve different types of optimization problems:

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

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.

The objective (x) = x2sin (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[3]. 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.

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.

[2] 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.

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

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.