Math 116, Spring 2014, Syllabus
Great papers in numerical analysis and computational algorithms

Alex Barnett

March 23, 2014

We meet in Kemeny 201 in the 2A slot (2-3:50pm Tu, Th, with intermission). You may and should bring laptops either to take notes into or do numerical experiments occasionally; if you don't we can share. Office hours are 2-3pm Wednesday, and 5-6pm Thursday; you can come to Math 56 office hours if no-one else is there, or by arrangement. X-hours are unused.

Email: `ahb` followed by `math` then dot `dartmouth.edu`.

Course website: `http://math.dartmouth.edu/~m116s14`

1 Overview

How are we able to compute the massive things that we can? How are we able to find eigenvalues of large matrices, solve PDEs with error 0.0000000001, optimize a function of thousands of variables, or solve linear systems with a million unknowns? Although faster hardware is part of the story, much of the credit goes to the invention (discovery?) of vastly more efficent numerical algorithms, say $O(NlogN)$ instead of $O(N^2)$.

This spring I will run Math 116 as a research-paper-based "journal club" where students will present some of the great numerical analysis and computational algorithms (either classic or recent). I first plan to lecture for two weeks to teach rapidly some key concepts in numerical analysis, and provide us with common langauge. Then we shift to student presentations of papers from a list (see below) of classic and modern papers, including some of the "Top Ten Algorithms of the 20th Century" (according to SIAM). Many of these algorithms are run millions of times per day worldwide, and have had a huge impact on science. We keep themes accessible: a focus on numerical linear algebra (since this hasn't been covered much in this department, yet is so important), also papers on Fourier methods, finite differencing, randomized algorithms. None of the papers are too technical: they require mostly linear algebra (rather than, say, functional analysis), and a enthusiasm to dig into algorithms even when subscripts get messy!

Your main workload will be presenting an appropriate chunk of your chosen paper during two 90-min lectures (ie 1 week of our 2A slot, with intermissions, and 10 mins at the end for discussion and filling feedback forms). You can use chalk, computer slides, handouts, or activities for the audience which teach them ideas, or a mixture (I encourage this). A large part of your presentation should be the main mathematical ideas/theory/algorithm, defining new objects clearly. Then I suggest you do computer implementation/experiments, either of a piece of the algorithm, or the whole thing (ask me what's reasonable). You should also try to answer:

Why is this algorithm better than other ways of solving the problem? What research areas in science use the algorithm? What impact has it had?

Each week one of the other students will be "scribe" and take as good notes as they can, and type them up in a few pages of LaTeX; the lecturer will check over them for corrections.

Your work (and how I assess you) will then be:

- Reading, understanding enough of, and presenting, your paper and numerical experiments, checking your scribe's writeup: 75%
- Being scribe for one of the other presentations: 15%
- Overall class participation, giving feedback, asking questions: 10%

Your workload will therefore be concentrated into the week before, and the week of, your presentation; most of the rest of the quarter you need only be an engaged class participant.

You do not have to do implement the algorithms you present, but I suggest you try this; none of them are too complicated to code. Why? It's fun, Matlab makes it pretty quick to do, and I and your peers are here to help. Presenting your code and its results could form a chunk of your 2nd (ie Thursday) lecture. I will consider it all part of your effort.

Choose your paper by **Tuesday April 1st**. If you don't get your first choice, rest assured they are all beautiful and interesting. I would like the papers to be presented roughly in historical order.

There is no final exam, although I may give summary lectures at the end.

2 Goals for this course

- Appreciating some of the highest-impact mathematical algorithms for numerical computation.
- Understand core ideas from many of these algorithms; breadth not depth.
- Learning at least one set of ideas/algorithm deeply by having to explain them to others.
- Improve communication skills: practice extracting the core ideas, presenting them as clearly as possible, learn from audience feedback.
- Learning to read research literature: knowing when to keep pushing, when to skim, when to seek help.
- Coding up some algorithms to see how they perform, numerical experimentation.
- Appreciate some current research directions in numerical mathematics.

3 Hints

1. This course is about skills any successful researcher needs: reading complicated stuff, extracting what you need, explaining to others, and implementing/testing it.
2. Start reading 2 weeks before, and look/ask around for textbooks, lecture notes, reviews (not just wikipedia!) to help you understand the paper and its context. Start writing your two lectures 1 week before.
3. Do a little practice writing clearly with chalk if you haven't done this much—press hard, write big.

4. Explain things to each other; explain them to me in office hrs. Keep a list of what you're stuck on. Don't worry if there's some bits of the paper you never quite get; focus on what matters. Narrative flow is important.

5. Don't get carried away with perfect computer presentations, although `beamer` is useful.

6. Start messing around with Matlab to get into the swing of things. Is there a command that already does your algorithm? (if so, well, you should still implement it; if not, why not?)

7. Think if there's a mathematical step/proof you can have *us* the audience do, give us the pieces and try it. More fun than watching you do it at the board.

4 The papers (in historical order)

Here's the papers to choose from; download them from our website. The shorter ones are no easier than the long ones; I expect the same amount of work. I also include guidance and thought-provoking questions to try and answer.

4.1 Courant–Friedrichs–Lewy 1928: Finite difference methods

One of the "Top Ten Algorithms", and the mainstay of how PDEs are solved. Elementary proofs (using just sums) of convergence from a time before functional analysis took over, and before electronic computers! (Even proves existence of Laplace BVP solution via the finite difference method.) Originally in German; we'll read the translation. You would want to have some background or interest in PDEs and real/functional analysis. Focus on I.4 which proves convergence for Laplace's equation, and II.2-3 which does same for the wave equation and gives the famous "CFL condition". Sections on biharmonic or 3D can be skipped. LeVeque's book is a modern resource. Provides motivation for solving linear systems. A little wave equation code would be fun and easy.

Questions:

- How do the convergence proofs in I.4 and II.3 use compactness?
- No mention of *rate* of convergence of the finite difference approximation is given—what is it for the stencils considered?
- In what situations are finite elements to be preferred over finite differencing? Boundary integral methods? Discuss.

4.2 Davidon, Fletcher–Powell 1963: quasi-Newton methods for non-linear optimization

Newton minimization methods in $\mathbb{R}^n$ requires a second-derivative (Hessian) matrix which is too expensive to compute; DFP (and the very similar BFGS) just need the first derivative, which make them some of the best optimization algorithms for smooth local minima for high n. You read the FP paper (you can skip Sec. 5 unless you don't code), but will need to read other more modern explanations to appreciate it. Eg Sec. 8.1 of the Nocedal–Wright optimization book (and Sec. 3.1 for approximate line search and Wolfe conditions). Also Fletcher's 1970 TJC paper. It would be very good to code this up and show it in say $\mathbb{R}^2$. W. C. Davidon, the originator of the idea, was a physicist ... and also broke into the FBI!

Questions:

- What is the origin and interpretation(s) of the inverse Hessian update formula?
- How do you get from the low-rank update of a matrix to a low-rank update of its inverse?
- Is the performance different from BFGS? (You may discuss/code BFGS too if you want.)

4.3 Cooley–Tukey 1965: Fast Fourier transform (FFT)

One of the most beautiful of the "Top Ten Algorithms", revolutionized signal processing. Other than basic summation over powers of the roots of unity, little background is needed. The paper is quite algebraic, so pictures will help. The wikipedia page on FFT explains a different (recursive) algorithm.

Questions:

- Why does bit-reversal work? How does it compare to the recursive algorithm?
- History: what did Gauss know about this?
- How do you do FFT for N prime?

4.4 Francis 1961–62: QR iteration for nonsymmetric matrix eigenvalue problem

This is one of the "jewels of numerical analysis" [NLA]. Part 1 presents theory, of which you should aim to follow most, whereas part 2 has implementation details which you need not present (especially the stuff designed for complex eigenvalues). The book [NLA] will be your friend: it also has the "practical" QR, as well as the convergence theorem; you may use this if you prefer it to the paper's version. The implementation should be fun; you can use complex arithmetic in Matlab.

Questions:

- Why is Hessenberg form used? Why can't triangular form be used?
- How do modern versions of the QR iteration decide on shifts and deflation?
- How much faster is today's laptop than a "Pegasus" of 1961?

4.5 Metropolis 1953, Hastings 1970: Markov chain Monte Carlo (MCMC)

Another Top Ten; this idea revolutionized computational statistics, physics, chemistry, etc. The project is for the more statistically-minded, although everyone should be able to understand it. You can skip Sec. 2.5 and Sec. 3. You will also want to read MacKay's 1996 Erice review, and demo the algorithm for sampling some pdf (eg the likelihood of some data given one or more parameters of a predictive model; ask me for examples if stuck).

Questions:

- Why is computing expectations of a pdf hard in high dimensions?
- How do you decide the variance of the average of a quantity along the chain?
- What modern techniques are there to reduce variance (mixing time) hence increase efficiency?

4.6 Saad–Schultz 1986: GMRES non-symmetric iterative solution of linear systems

This method has become the workhorse for solving large square linear systems that have sufficiently "good" spectra, with unknowns up to the billions. This paper builds on basic material in part II of [NLA], and is also covered by Ch. 33 and 35 [NLA]. Use these to help with your best explanation of the algorithm. You can skip the restarted versions GMRES(m), and don't focus on the update of the QR at each step; $O(k^2)$ is fine. It would be good to study alongside it, and check with your own matrices, some part of a beautiful paper on GMRES convergence rate,

T. A. Driscoll, K.-C. Toh, and L. N. Trefethen, "From potential theory for matrix iterations in six steps," SIAM Review, **40** (3), 547–578 (1998).

Questions:

- On p.858 why does $H_k = V_k^T A V_k$ hold for the subdiagonal entries?
- How fast does GMRES converge? (You can use [NLA] or the other paper for this since the original paper is brief.) What are the effects of condition number, spectrum, and non-normality?
- Comment on the bizarre horizonal axis in the results figures.

4.7 Greengard–Rokhlin 1987: Fast Multipole Method (FMM)

Another Top-Ten. This is an influential paper, the first to show how to guarantee controlled accuracy in fast summation of the potential interactions of N particles, in $O(N)$ time. Resources: Lec. 15–16 of my Math 126, Winter 2012 (online; note that my monopole convention has the opposite sign); Beatson notes. Complex variables play a fun role in proofs. You don't have to go through all proofs (eg skip Lemma 2.4), and can skip the various boundary conditions in Sec. 4. For coding, showing the low rank of interaction of particles in well-separated boxes, and evaluating via a single source-to-multipole, and multipole evaluation at targets would be a start, then a single-level scheme.

Questions:

- What asymptotic complexity would a 1-level FMM have?
- What is the interaction list and why is it needed?
- What is the difference between an FMM and a "tree code"?

4.8 Halko–Martinsson–Tropp 2011: Randomized algorithms for matrix factorization

This reviews modern methods for computing eg SVD of large matrices; such methods are becoming popular in an age of big data where communication (rather than flops) limits CPU speed. The review is 70 pages although very well written, so I'd expect you to focus on Sec. 1 (basic idea), explaining only a theorem or two, then trying out algorithms from Sec. 4 and 5 and showing us results on some large matrices (eg for PCA?). This can be a more practical rather than proof-oriented project.

Questions:

- How big does a dense matrix have to be before the randomized SVD becomes faster than the standard dense SVD?

- Same for large sparse matrix. In both cases what does the answer depend on?

4.9 Other papers under consideration

- J.-P. Berrut and L.N. Trefethen, "Barycentric Lagrange Interpolation," SIAM Review, **46** (3), 501–517 (2044).
- Maximization of a Linear Function of Variables Subject to Linear Inequalities, by G.B. Dantzig, Activity Analysis of Production and Allocation, pp. 339-347. Cowles Commission Monograph No. 13. John Wiley & Sons, Inc., New York, N. Y.; Chapman & Hall, Ltd., London, 1951.

5 References

This course is based upon Trefethen's from Oxford, 1993, and numerous other incarnations such as:
Gene Golub: `http://www.stanford.edu/class/cme324/classics/`
Scott MacLachlan: `http://neumann.math.tufts.edu/math250_S12`

5.1 Useful books and papers

[NLA] L. N. Trefethen and D. Bau, Numerical Linear Algebra (SIAM, 1997). Beautiful accessible introduction.
[MC] G. H. Golub and C. F. Van Loan, Matrix Computations, 4th Edition (John Hopkins, 2012). Pretty much the bible for numerical linear algebra.
[NA] R. Kress, Numerical Analysis, Graduate Texts in Mathematics 181 (Springer, 1998). Clear, concise, analysis-leaning overview of the whole subject. Excellent for quadrature and integral equations, poorer for linear systems and finite difference methods.
[FD] R. J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations (SIAM, 2007).
[NW] J. Nocedal and S. J. Wright, Numerical Optimization, (Springer Verlag, 1999).