Overview

Below are some topics to choose from. Each topic consists of some mathematical problem of interest with interesting numerical properties; some of them also involve applications. If you have an interest in a certain field, there may be room to adjust an existing topic to include it. Some topics here are broad and could be taken in various directions (do a little research if one sounds interesting). The actual project could (and in most cases, probably should) just zoom in on one facet of a topic.

Regardless of what you intend to select, you should discuss your thoughts with me at some point, preferably before topics are due but certainly before they are finalized. It is possible that you do not end up with your first choice.

The area of the topic is labeled in blue and some ‘search keywords’ to consider when researching the topic are in purple. Suggested background is listed in red. For most topics, this background is not required but is helpful (otherwise you will need to learn as you go). I have starred topics that I think are challenging without background in the subject.1

When available, relevant chapters in 'the book' (Ascher & Greif) are noted.

Linear algebra topics

Numerical linear algebra is an expansive sub-field of numerical analysis, and has developed greatly in the last few decades, driven by the need for algorithms that can handle large data sets and theory that can provide insight into the structure of that data.

**Singular Value Decomposition: (Book: Chapter 4.4 (intro) and 8 (details))**

(Area: matrix decomposition (like LU); data analysis)

(Keywords: Non-linear optimization; linear algebra)

(Background: Eigenvalues/eigenvectors; this is a challenging topic that requires a good amount of linear algebra, plus a willingness to dive into the nasty details of numerical linear algebra.)

A matrix with a full set of eigenvalues/vectors can be diagonalized, which gives it nice properties. When it does not, we must instead use the 'singular value decomposition',

1For a few topics, background is good for context but not essential to the problem (for instance: we did not need to use much ODE theory to solve them numerically, but the topic is more meaningful if you have studied them before).
which extends this notion. The SVD is a powerful tool that lets one extract the 'most important' information from a high-dimensional set of data - for instance, compressing an image by throwing away unneeded data.

Remarkably, a number of practical problems can be solved, more or less, by just computing the SVD of the right representation of the data (e.g. image compression, ‘principal component analysis’).

The main disadvantage of this topic is that computing the SVD is technical (it involves several techniques that are messier than simple LU decomposition), and the theory is non-trivial. There are, however, a few cheap ways to compute it.

**Least squares (QR):** *(Book: Chapter 6)*

(Area: Optimization; matrix decomposition (like LU))

(Keywords: Givens rotation / Householder transformation; QR factorization; Gram-Schmidt)

(Background: Linear algebra from class)

We saw that least-squares problems can be solved using the normal equations. This is not the 'best' way to do it. The QR factorization is a more robust technique, and perhaps the most important factorization (even more than LU). Least-squares is used for obtaining 'best-fits' to data, which has innumerable applications.

**Ill-conditioned matrices, condition estimation:**

(Source: Golub and Van Loan, *Matrix computations*, p128-130.)

(Area: Linear algebra; error analysis for linear algebra)

(Keywords: Condition number estimation; iterative refinement)

(Background: none)

How can we tell if a matrix $A$ is ill-conditioned? The trouble is, we need the inverse. But if $A$ is ill-conditioned then trying to calculate this (and related quantities) is itself an ill-conditioned problem.

Besides, we want to check if $A$ is nice before doing real computations - we want something quick and approximate. The exact value of $\kappa(A)$ is almost never important, just its order of magnitude. The way forward is to estimate the condition number $\kappa(A)$ by 'testing' the matrix in a clever way. There are a number of techniques to do so, which exploit some linear algebra tricks and heuristics to balance effort and quality of approximation.

You may also want to consider a bit about ill-conditioned systems and how to deal with them. (e.g. iterative refinement) This involves delving a bit further into error analysis.

**Conjugate gradient / Krylov subspace methods:** *(Book: Chapter 7)*

(Area: Linear algebra)

(Keywords: Conjugate gradient; Krylov subspace; sometimes Lanczos/Arnoldi iteration; iterative methods)

(Background: Linear algebra (iterative methods from class).)

We saw in a homework that the Conjugate Gradient method works, but used it as a mysterious black box. You could delve into the details and understand why it is such a useful method. There is substantial theory involved (the ‘Krylov subspace’, which is the subspace spanned by $b, Ab, \cdots, A^k b$) in order to properly derive the method and understand the behavior of the error. There are also some tricks to make CG even faster (‘preconditioning’ to prepare the system in advance).
**Note:** There are other, more sophisticated methods like CG that are probably beyond the scope of a project like MINRES/GMRES/BiCG. You’ll find them mentioned often alongside CG, as they work in the same theoretical framework.

### Differential equations

**Stiff systems of ODEs: (Book: Chapter 16.5)**
- **(Area:** ODEs)
- **(Keywords:** Stiffness, absolute stability)
- **(Background:** ODEs (what we cover in class; we will briefly address stiffness)

When a physical system has multiple time scales - for instance, a chemical reaction with one part that goes quite fast, and another that goes quite slow - numerical methods for ODEs can have trouble. A **stiff system** is a system of ODEs that requires a different sort of method. Stiff systems arise often and must be handled with care, since they will slow down simple methods to the point of being unusable.

The catch is that methods for stiff systems require more computational work - typically involving the solution of a non-linear system at each step. This is, however, better than waiting a month for **ode45** to finish computing, so it has to be done.

The study of stiff systems leads to interesting connections to ODE theory as well - identifying stiff systems requires learning some intuition both how ODEs behave and the corresponding numerical methods.

**Elliptic PDEs: (Book: Scattered; Example 7.1 on p. 168 is a good starting point)**
- **(Area:** PDEs, but really linear algebra)
- **(Keywords:** Laplace’s equation; Poisson’s equation, SOR, Elliptic equations)
- **(Background:** linear algebra (what was done in class). A bit of ODE background helps but is not required.)

Elliptic PDEs are one of the main classes of PDEs. The most famous example is Laplace’s equation, which describes diverse phenomena from the flow of a fluid past an object (imagine a wind tunnel) to the wavefunction of an electron. This system is sparse and often banded (almost tridiagonal), and can be solved in a variety of ways. Efficiency and accuracy of of great concern, so all sorts of fast/reliable/general methods have been developed.

In theory, this is a PDE problem but from a numerical perspective, it is a linear algebra problem. The system can be solved directly (using some LU-like algorithms), or using iterative methods (SOR, or successive over-relaxation, is an extension of Gauss-Seidel that can be used). The tri-diagonal system from the homework is an example.

**Parabolic PDEs (finite differences): (Book: N/A)**
- **(Area:** Linear algebra; ODEs)
- **(Keywords:** Heat equation; parabolic equations; method of lines, finite difference methods (Forward time, centered space; Crank-Nicholson))
- **(Background:** ODEs recommended; linear algebra from class. PDEs not required but helpful.)

---

\(^2\)This is really Schroedinger’s equation, but it is similar in some cases, and the numerical techniques transfer
Solving partial differential equations with time dependence can be done by replacing the derivatives with finite differences. This leads to a linear algebra problem to ‘advance’ the system a small time step \( \Delta t \). Methods must be designed to do this stably and accurately; the concerns are similar to what we cover for ODEs.

The computations amount to solving linear algebra problems (same as for elliptic PDEs, but a bit simpler), or solving a system of ODEs, or a mix of the two. Efficiency in the linear algebra part really matters here, since one needs to solve an \( Ax = b \) problem at each time step, and the size of the problem can be quite large.

Questions of numerical stability are interesting here - there are close analogies to the corresponding notions for ODE methods and subtleties that anyone that needs to solve PDEs numerically should know.

**Boundary value problems:** *(Book: Chapter 16.7)*

*(Area: ODEs, linear algebra)*

*(Keywords: Boundary value problems; shooting; finite differences)*

*(Background: Knowledge of ODEs are helpful.)*

In class, we study ‘initial value problems’, where a system is advanced in time over a series of steps. A **boundary value problem** specifies conditions at more than one location, so we cannot simply follow the solution from a starting point.

There are several techniques to consider. Shooting uses initial value problems and root-finding (as covered in class) to match the boundary conditions. Finite difference methods discretize the domain and solve for the whole solution at once, turning the process into a linear algebra problem.

In either case, there are subtleties, as the structure of a boundary value problem is more complicated than a simple initial value problem. Shooting can behave badly; finite differences are strongly influenced by boundary conditions, and some problems are just difficult to solve accurately.

Notable examples include the *Van der pol oscillator*, *traveling wave solutions* for differential equations, the shape of a solid (e.g. deformation of a bridge) and more.

If you are familiar with eigenvalue problems for PDEs (like the heat equation): you could look at calculating eigenvalues numerically (which amounts to finding eigenvalues of the discretized system).

**Simulating ‘Particle’ systems:** *(Book: N/A)*

*(Area: differential equations)*

*(Keywords: point vortices; three body problems; chaotic systems (of ODEs))*

*(Background: ODEs are helpful)*

A vortex in a fluid moves with the flow, but also exerts an influence (as it swirls around) that affects other vortices. They interact and produce complicated dynamics (including chaotic motion). Such a system can be modeled by a coupled system of ODEs (one for each vortex); simulation of the system requires some effort because the amount of work increases dramatically with the number of objects and the complex dynamics make accurate (and reliable) calculation difficult.
There are a number of related problems to this, including the classical three body problem (for orbiting planets) and particles sinking in a fluid.

**Stochastic differential equations:**
(Area: differential equations, probability)
(Keywords: Euler-Maruyama method; SDE (stochastic differential equation))
(Background: ODEs are helpful, PDEs even more so. Some probability required.)

What happens when we add randomness to a differential equation that depends on time? The ‘solution’ is now not just a single trajectory in time, but a family of possibilities, depending on the result of the randomness. The fluctuation of stock prices, populations, floating particles (like a speck of dust in air) and much more can be modeled with SDEs. The random processes are challenging to handle numerically and require careful modification of the standard methods (see the Euler-Maruyama method for the simplest case). Error analysis is much more nuanced as well, since we no longer have one solution that always behaves the same way.

**Other topics**

**Singular integrals:**
(Area: Integration; calculus; physics)
(Keywords: Double exponential / tanh rule, singular integral, improper integrals )
(Background: basic calculus)
Ill-behaved functions are hard to integrate numerically. Functions with singularities may be integrable in theory, but cause trouble for basic numerical methods. Computing

\[
\int_0^1 \log(x) \log(1-x) \, dx = 2 - \frac{\pi^2}{6}
\]

is not so easy. As it turns out, singular integrals show up often in mathematics - for instance, calculating forces between objects (when repulsive forces diverge as the objects get close together). Developing methods for integrating singular functions requires some finesse and a number of clever tricks. The right method and change of variables can turn a very nasty problem into a very nice one.

**Spectral differentiation:** (Book: Chapter 14.5)
(Area: Fourier analysis, linear algebra, differential equations)
(Keywords: Differentiation matrix; fourier transform; FFT; Chebshev differentiation)
(Background: Fourier series and/or Fourier transform)
A derivative corresponds to multiplication in Fourier space. Multiplication is much easier - and much less error-prone - than differentiation. One strategy to differentiate is to Fourier transform the function, do some trivial multiplication and then transform back. This remarkable trick is the foundation for modern methods to solve differential equations (where derivatives, of course, must be dealt with) and much more.

The efficiency of the method is due to the Fast Fourier transform, which lets the transformation to and from Fourier space be done quickly. **Note that** with this topic, you can

---

\(^3\)Example from *Numerical Recipes, 3rd. Edition* by Press et. al.; Chapter 4.5 is a good resource if you want to learn more about this
focus on the application to differentiation rather than the FFT itself; there are simpler ways to do the transforms.

**DFT/FFT/Trig interpolation:** *(Book: Chapter 13)*

*(Area: Fourier analysis; signal processing; integrals)*

*(Keywords: Discrete fourier transform; fast fourier transform)*

*(Background: Fourier series/transform is helpful but not necessary. We might cover some of this in class.)*

The discrete Fourier transform takes a discrete set of data and converts it into 'Fourier space', decomposing it into its frequencies (say, for instance, a radio signal converted into its spectrum of (literal) sound frequencies). With the discovery\(^4\) of the Fast Fourier transform - an almost magical algorithm for computing the DFT quickly - Fourier-based methods found their way into numerical computing. It is used heavily in signal processing (e.g. digital filters) and show up in all sorts of places (solving differential equations, multiplying two numbers with \(n\) digits in \(O(n \log n)\) time).

Trigonometric interpolation refers to using sines and cosines (\(\sin \theta, \sin 2\theta, \cdots\)) to interpolate data rather than polynomials. This is closely related to Fourier series and the discrete Fourier transform. Unlike high-degree polynomials, trig. interpolants do fine for large \(n\), and have all sorts of interesting properties.

**Splines:** *(Book: Chapter 11)*

*(Area: Interpolation, approximation)*

*(Keywords: interpolation)*

*(Background: interpolation, linear algebra)*

We briefly addressed ‘piecewise interpolation’. The **cubic spline** is a piecewise cubic interpolant that matches function values and derivatives. It is an popular way to represent a curve using a discrete set of points (much more so than a high degree polynomial!). **Bezier curves** and other variants are used in graphics to draw curves (e.g. the pen tool in Photoshop). There are subtleties to it - there is no one perfect scheme, and the details must be worked out to get the properties you want (how smooth does it have to be? Is it important to ensure that it ‘looks like’ it has the right shape?).

**Gauss-Newton method / Unconstrained minimization:** *(Book: Chapter 9.2)*

*(Area: Minimization; Newton’s method; least squares)*

*(Keywords: Unconstrained minimization/optimization; descent methods, Gauss-Newton)*

*(Background: none)*

A problem of obvious importance is to minimize a nonlinear function \(F(x) : \mathbb{R}^n \to \mathbb{R}\) of several variables. This can be done using Newton’s method to find a zero of the gradient \(\nabla F\), but that has some disadvantages. A variation is the ‘Gauss-Newton’ method, which solves a least-squares problem at each step. This algorithm can perform better than Newton.

More generally, you could look at other fundamentals of non-linear minimization: for instance, the non-linear version of steepest descent. The trick with these methods is that we want the iteration to go fast, but also want it to guarantee some amount of progress at each step. Gauss-Newton and steepest descent both do this (but Newton’s method does not).

\(^4\)Technically, re-discovery; the algorithm was known to Gauss in the 1850s, but it was far before anyone realized it had any applications so the method was forgotten until computers.