APPM 4600 — Project list

Iterative solvers and conditioning
You have learned about the condition number of a square matrix in class and how it impacts how accurately one can invert the matrix. High condition number can lead to a loss of some to all digits of accuracy. In this project, you will learn about iterative solvers for linear systems and how the condition number can impact the convergence. For your extension you will explore how a matrix preconditioner can be used to improve the conditioning of a ill-conditioned problem. This project will require you to employ software to solve these ill-conditioned problem and will require little mathematical derivation. If the coding side of numerics is not for you you should not chose this project.

Conjugate Gradient Method
In class we talked about the steepest descent algorithm and its connection to solving a linear system. When the matrix is symmetric, the same derivation with some extra bells yeilds the Conjugate Gradient method. This method has the advantage that it converges faster (in general) because it keeps track of previous directions traveled. In this project, you will derive the conjugate gradient method and compare its performance with that of steepest descent for a collection of problems. For the independent portion of the project, you will explore other iterative solvers for linear systems which can be applied to more general problems. For this project you are allowed to use software packages but you must be able to explain the algorithms in addition to having a working knowledge of the limitiations and advantages of each method.

The Fast Fourier Transform
In class, we talked about trigonometric interpolation. It turns out that when this interpolation involves uniformly distributed points, the cost of doing the interpolation can be reduced from $O(n^2)$ to $O(nlog(n))$. The technique for doing this is called the fast Fourier transform (FFT) and it exploits the fact that you can write the evaluation as beautifully nested series that all look the same but are scaled by a constant. In this project you will actually derive it and investigate its asymptotic behavior. Next you will learn the non-uniform FFT. What are the challenges of this method? How does it perform relative to the uniform FFT? What are recent developments on these methods?

The Minimax Method
For many applications a minimax approximation is desirable because the error is uniformly bounded in the entire interval of interest. In this project, you will explore the minimax approximation, deriving it and the associated conditions. Then, you will find the link between minimax approximations and the Chebychev polynomials. Additionally you can explore with prepackaged software the different in performance minimax approximations and the least squares approxima tions. You are free to come up with ideas for the independent part of the project.

Orthogonal Polynomials
Orthogonal polynomials have lots of fun properties. One of the lesser used properties is that the roots are nested. Specifically, let $P_n(x)$ denote an orthogonal polynomial on $[a, b]$ and let $x_{n,i}$ denote the roots of $P_n(x)$. Then, the roots of $P_{n+1}(x)$ satisfy $x_{n,i-1} < x_{n+1,i} < x_{n,i}$. In this project, you will prove this property about the roots and derive an efficient and robust algorithm for evaluating the roots of arbitrary orthogonal polynomials. To do so, you will make use of the three term

recurrence relation and other properties of orthogonal polynomials. One possible option for the independent part of the project includes developing robust and efficient techniques for evaluating the associated quadrature weights, and building from your algorithm for general weight functions to get an efficient least squares approximation routine. Another possible option is to analytically solve a 2-point boundary value problem by representing solutions as expansions in an orthogonal polynomial basis. This is the starting point for developing a "spectral" numerical method.

Sensitivity of Discrete Least Squares
In many applications the data is noisy; i.e. the data is not exactly the data you want to fit. In this project, your task is to explore the sensitivity of the discrete least squares approximation to the noise in the data. Does the accuracy depend on the number of data points and/or the size of the noise? Are there other things that come into play? Find theory in the literature to support your findings. Possible options for the independent part include techniques for pre-processing data to make it less noisy and least squares techniques that make it more robust to noise and outliers (e.g. smoothing).

Quadrature for Weakly Singular Functions
One of the take aways from Calculus II is that while we have a collection of tools for integrating functions, it is not possible for us to integrate all functions that are integrable. In class, we learned techniques for approximating integrals by strategically weighted Reimann sums. We called this quadrature. While this increases the range of functions we can "integrate," there are still functions that are integrable which you do not have the tools to approximate the integral of. In this project, you will investigate techniques for approximating the integral of weakly singular functions.

For this project, you will consider the problem of approximating the definite integral of a function $f(x)$ in the interval $[a, b]$ with an integrable singularity at $x = a$ (the improper integral converges). Assume you know what kind of singularity it is, e.g. you can write $f(x) = \frac{q(x)}{(x-a)^\alpha} + r(x)$ for $\alpha \in (0, 1)$, and $q(x), r(x)$ are analytic. You will derive an algorithm which constructs quadrature for approximating the integrals of these weakly singular functions. You will explore the performance of these techniques for a variety of different functions. In the independent poriton of the project, you can explore extensions of this idea to other singular (or hypersingular) integral or alternative techniques for approximating integrals of weakly singular functions including corrected trapezoidal rules, singularity subtraction and Taylor expansion based techniques.

Sparse solvers
Many applications such as constructing cubic splines and solving differential equations using the finite element method result in a system of linear equations that can be represented by a sparse matrix. A matrix A is called sparse if the number of zero entries *significantly* out number of non-zero entries. For example, the matrix that needs to be inverted to create cubic splines is tridiagonal. Thus for a large number (i.e. $>> 3$) interpolation nodes, the matrix that needs to be inverted is sparse. In this project, you will derive an inversion technique for tridiagonal systems whose computational cost scales linearly with respect to the size of the system. This algorithm is called the Thomas algorithm. Next, you will investigate inversion techniques for general sparse linear systems; i.e. sparse systems where you do not know apriori the sparisty pattern. These solvers a LU solvers which pick pivots in a manner such that the fill in of the factorization is minimized. These methods are call nested dissection or multi-frontal methods. You will explore some of the different pivoting techniques and investigate where these methods are used in practice and their

stability.

Regularized Least Squares
Regularized least squares is a family of methods that adds terms to the least squares formulations to constrain the solution. Regularization can helps users incorporate goals that we want the resulting approximating function to have such as the size of the coefficients, sparsity (how many are non-zero), smoothness of the model, etc. In this project, you will explore the least squares regularization techniques used in ridge regression and Tikhonov regularization. How do the different techniques effect the resulting least squares problem in terms of normal equations and other linear algebra techniques? In this project you will explore how these techniques can be used to minimize the effects of noise and other factors on the resulting regression lines. Possible options for the independent direction of this project include generalizations of Tikhonov, L_1 regularization (e.g. LASSO), continuous regularized least squares, and smooth spline approximation (Reproducing Kernel Hilbert Spaces).

Rank revealing QR factorization
In class, we talked about the QR factorization. In many applications, it useful to create a low rank factorization which approximates a matrix to a user prescribed accuracy. This is useful, for example, when the low rank factorization provides a reasonable approximation to the original matrix, with a fraction of the cost when storing or computing with it. One of the most popular techniques for doing this is the rank revealing QR factorization. In fact, it is becoming a driving force in machine learning and reduced order modeling. In this project, you will learn about QR factorizations with pivoting. For the independent directions, you can consider a randomized variant of the singular value decomposition (SVD), and/or apply your low rank approximation methods (QR or SVD) to a real-world problem where the data can be compressed/approximated by such methods.

Newton Directions in Descent Methods for Optimization
In class, we discussed the steepest descent method applied to problem of finding roots of some $F : \mathbb{R}^n \to \mathbb{R}^m$. After re-writing it in terms of finding the minimum for the quadratic $f : \mathbb{R}^n \to \mathbb{R}$ defined as $f(x) = \sum_{j=1}^{m} F_j(x)^2 = ||F(x)||_2^2$, we derived the method with an iteration of the form $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$. The direction $-\nabla f(x_k)$ at step k is a specific instance of a descent direction (the "steepest") within a more general class of descent directions. We can choose α_k in a variety of ways, but the method can be slow on difficult problems. Another important descent direction is the Newton direction, which has a natural unit step length. In this project, you will derive the Newton direction $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$ and associated method, and numerically verify your implementation on a root finding problem. You will also explore some of the quasi versions of this method which address the inefficiencies with inverting the Hessian matrix. For the independent directions, you may consider variations when the Hessian is not positive definite, and/or apply your implementations to a different optimization problem of interest to you, including data-driven applications.