Numerical Analysis: Algebra and Approximation

George D. Torres

Fall 2019
Prof. Björn Engquist

1 Introduction **2**

2 Linear Solvers and Matrix Problems **3**
2.1 Background in Linear Algebra . . . 3
2.2 Perturbations of linear systems . . . 3
2.3 Gaussian Elimination . . . 5
2.4 The Cholesky Decomposition . . . 6
2.5 Iterative Methods . . . 7
2.6 Eigenvalue problems . . . 13
2.7 Singular Value Decomposition . . . 16

3 Nonlinear Equations **18**
3.1 Fixed point iteration . . . 18
3.2 Secant Method & Quasi-Newton method . . . 19

4 Optimization **20**
4.1 Unconstrained Minimization . . . 20
4.2 Constrained Optimization . . . 21
4.3 Linear Programming . . . 22
4.4 Nonlinear Constrained Optimization . . . 23

5 Interpolation and Approximation **24**
5.1 Lagrange Polynomial interpolation . . . 24
5.2 Piecewise Interpolation . . . 25
5.3 Hermite-Birkhoff Interpolation . . . 27
5.4 Splines . . . 28
5.5 Application to Numerical Differentiation . . . 29
5.6 Deep Learning . . . 30
5.7 Discrete approximation . . . 31
5.8 Orthogonality and L^2 approximations . . . 33

6 Numerical Integration **36**
6.1 Quadrature . . . 36
6.2 Richardson Extrapolation . . . 37
6.3 Singular Integrals . . . 37
6.4 Higher Dimensions & Monte Carlo Methods . . . 38

Last updated December 13, 2019

1. Introduction

❖

(*from the syllabus*)

This course covers fundamental topics in numerical solutions of linear and nonlinear algebraic equations and different aspects of approximation techniques will be covered. Modern development and material that is still relevant for numerical solutions of differential equations will be stressed. This is a stand-alone course but is also useful as preparation for the related spring course. Topics covered:

- Basic concepts in numerical analysis.
- Numerical linear algebra - including direct and modern iterative methods. Techniques for eigenvalues and singular values. Introduction to fast algorithms such as the fast multipole method (FMM) and to distributed computing.
- Numerical methods of solving of systems of nonlinear equations, including quasi-Newton methods.
- Optimization and constrained optimization.
- Machine learning as an application of linear algebra, optimization, and approximation.
- Interpolation and approximation, including orthogonal polynomials, FFT, wavelets, splines, and NURBS.
- Numerical integration and differentiation, including Monte Carlo methods.

The primary textbook used for this class is *Numerical Mathematics*, by Quarteroni et. al. (Texts in Applied Mathematics 37, Springer Verlag).

2. Linear Solvers and Matrix Problems

❖

Lecture 9/4

Suppose we want to solve a system of linear equations $Ax = b$. The direct method for solving this system is Gaussian elimination (or more specialized methods that use properties of A). Another type of method is the iterative type. Examples of these are multi-grid and the Krylov subspace method. There are also fast methods that have lower complexity that will solve the problem to a certain accuracy. A related method is the least squares method, which roughly seeks to minimize the norm of $Ax - b$.

Three characteristics in a linear solver that we seek are well-posed-ness (i.e. there is an answer), uniqueness of the solution (i.e. A is nonsingular), and robustness to perturbations.

2.1 Background in Linear Algebra ❖

Definition 2.1. A square matrix A is called *positive definite* if, for all vectors x, we have $x^T Ax \geq cx^T x$ for some constant $c > 0$. It is called *non-singular* if one of the following equivalent statements hold:

- $Ax = b$ has a unique solution.
- A^{-1} exists.
- $\det(A) \neq 0$.
- A has no zero eigenvalue (i.e. $0 \notin \sigma(A)$).

It is called *singular* if it is not non-singular.

Given a real or complex matrix A and a nonsingular matrix S, a similarity transformation is $B = SAS^{-1}$. A property of such a transformation is that $\lambda \in \sigma(B) \iff \lambda \in \sigma(A)$. The associated eigenvectors are changed by the transformation S. Using similarity, there are various ways to represent a matrix:

- $A = LU$, where L is lower-triangular and U is upper triangular. If A is symmetric and real, then $U = L^T$. In this case, it is called a Cholesky decomposition.
- $A = QR$, where Q is orthogonal and R is upper triangular.
- $U^* AV = \Sigma$, where U and V are unitary (or orthogonal if real). This is the singular value decomposition (SVD). The matrix Σ is zero except for a square diagonal block whose diagonals (the singular values) are nonnegative.

2.2 Perturbations of linear systems ❖

An important theorem regarding perturbations of linear systems is the following *a priori* estimate:

Theorem 2.2. *Suppose we have a linear system $Ax = b$ for A nonsingular. Then consider the perturbed system $(A + \delta A)(x + \delta x) = b + \delta b$, where $\|A^{-1}\| \cdot \|\delta A\| < 1$. Then we have the following estimate:*

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{K(A)}{1 - \frac{K(A)\|\delta A\|}{\|A\|}} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right)$$

where $K(A) = \|A\| \cdot \|A^{-1}\|$ is the condition number.

Here we are using the operator norm $\|A\| = \sup_{\|x\|=1}\|Ax\|$ and any p-norm on on vectors. For certain values of p, one can show these take the form:

$$\|A\|_1 = \max_j \left(\sum_j |A_{ij}|\right) \qquad \text{(max column sum)}$$

$$\|A\|_2 = \sigma_{max}(A) \qquad \text{(max singular value)}$$

$$\|A\|_\infty \max_i \left(\sum_j |A_{ij}|\right) \qquad \text{(max row sum)}$$

Corollary 2.3. *Suppose we only perturb b (i.e. $\delta A = 0$). Then we have the following estimate:*

$$\frac{\|\delta x\|}{\|x\|} \leq K(A)\frac{\|\delta b\|}{\|b\|}$$

Proof:

Taking the perturbed system $A(x + \delta x) = b + \delta b$ and multiplying by A^{-1}, we get:

$$x + \delta x = A^{-1}b + A^{-1}\delta b \Longrightarrow \delta x = A^{-1}\delta b$$

because $Ax = b$. Then by the multiplicative triangle inequality:

$$\|\delta x\| \leq \|A^{-1}\| \cdot \|\delta b\| \Longrightarrow \frac{\|\delta x\|}{\|x\|} \leq \|A^{-1}\|\frac{\|\delta b\|}{\|b\|}$$

Now using $\|b\| \leq \|A\|\|x\|$, we get:

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\delta b\|}{\|b\|}$$

and the result follows.

□

We would like to use the same idea to prove Theorem 2.2, which will require the following

Lemma 2.4. *Given a matrix B such that $I + B$ is invertible and $\|B\| < 1$, then $\|(I + B)^{-1}\| \leq \frac{1}{1-\|B\|}$.*

Proof:

We can write:

$$(I + B)^{-1} = I - B + B^2 - B^3 + ...$$

which is the Neumann series. Taking the norm of both sides and applying the triangle inequality produces the result.

□

Proof of Theorem 2.2:

We start with the perturbed equation $(A + \delta A)(x + \delta x) = b + \delta b$ and multiply both sides by the matrix inverses and re-arrange to get:

$$-\delta x = (I - A^{-1}\delta A)^{-1}A^{-1}(\delta b - \delta Ax)$$

We use the lemma with $B = A^{-1}\delta A$ to get:

$$\|\delta x\| \leq \frac{\|A^{-1}\|}{1 + \|A^{-1}\|\|\delta A\|}(\|b\| + \|\delta A\|\|x\|)$$

Divide both sides by $\|x\|$ and use $\|x\| \geq \frac{\|b\|}{\|A\|}$ to obtain the result.

□

Lecture 9/9

Definition 2.5. The *Frobeneus* norm of a matrix is:

$$\|A\|_F = \sqrt{\sum_i \sum_j |A_{ij}|^2}$$

2.3 Gaussian Elimination ❖

The most well-known algorithm for solving a non-singular equation $Ax = b$ is Gaussian Elimination. It consists of eliminating more and more variables from each successive row of A. This process is equivalent to LU factorization. This decomposition also has the same complexity as Gaussian Elimination (which is $\mathcal{O}(n^3)$). Once we have computed the LU decomposition, we then solve:

$$\begin{Bmatrix} Ly = b \\ Ux = y \end{Bmatrix}$$

This is easy since L and U are triangular, assuming A is symmetric (use the method of back-substitution). This is also known as the Cholesky decomposition method.

Question: Can we do better than Gaussian Elimination, perhaps at the cost of precision? Since there are n^2 entries of A, then certainly any algorithm must be $\mathcal{O}(n^p)$ for $p \geq 2$.

- The Strassen algorithm is an example of a linear solving algorithm that gives an exact answer with $\mathcal{O}(n^{2.8})$. However, it has issues with error propagation and the constant in front of $n^{2.8}$ is large. It is a hierarchical algorithm that uses block forms of matrices.
- The Coppersmith-Winograd algorithm has $\mathcal{O}(n^{2.4})$ but is computationally not practical because of a large leading constant (more of theoretical value).

An important remark is that A being nonsingular doesn't necessarily imply that Gaussian Elimination or LU decomposition will work. One might first need to perform row or column pivoting. For example:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

This requires a row pivot (switching of the two rows). Even if the first entry of A is nonzero, but is very small, issues with precision can arise. Therefore it is advisable to pivot the row with the largest first entry (in absolute value) to the top. Alternatively, one could pivot the column with the largest first entry to the first column. *Total pivoting* is the process of exchanging rows and columns to bring the largest entry of the matrix A to be the first entry. This results in even smaller error propagation. Row and column pivots can be written in matrix form:

$$PA \tag{Row pivot}$$

$$AP \tag{Column pivot}$$

where P is a permuation matrix.

Lecture 9/11

2.3.1 Direct methods for special matrices

Linear solving can be improved from Gaussian Elimination if the matrix A has special properties that can be exploited. These types of matrices include sparse matrices, block matrices, and positive definite matrices. For example, with block matrices, methods of distributed computing (parallel processing) are used. With sparse matrices, we keep track of only the nonzero entries of the matrix.

Example 2.6. Consider the process of Gaussian Elimination parallelized. We break up the rows into subsets by assigning the first m rows to the first processor, the second m rows to the second processor, and so on. The pivot element is calculated across all subsets and then elimination is performed on each processor for its respective rows.

$$\begin{bmatrix} ---------- \text{Processor 1} ---------- \\ ---------- \text{Processor 2} ---------- \\ ---------- \text{Processor 3} ---------- \\ \vdots \\ ---------- \text{Processor N} ---------- \end{bmatrix}$$

The problem with the way these processors are arranged is that eventually Processor 1 will become idle, wasting resources. So a better way to assign tasks to processors is to interleave the rows assigned to each processor. In other words, the rows assigned to Processor 1 are $\{1, m+1, 2m+1, ...\}$, the rows assigned to Processor 2 are $\{2, m+2, 2m+2, ...\}$ and so on.

Lecture 9/18

2.4 The Cholesky Decomposition ❖

Recall the Cholesky decomposition was an LU factorization of a matrix A when it is symmetric and positive definite. Once we find such a decomposition, the linear system $Ax = b$ can be solved by back and forward substitution.

Proposition 2.7. *Let A be a positive definite matrix. Then:*

1. *A is nonsingular.*
2. *The diagonals A_{ii} are positive. If A is only positive semidefinite, then it A_{ii} is nonnegative.*
3. *If A is positive semidefinite but not positive definite, then it is singular.*

Theorem 2.8. *Let $A \in \mathbb{R}^{n\times n}$ be symmetric and positive definite. Then there exists a unique upper triangular matrix H with positive diagonals such that $A = H^T H$. This factorization is called the Cholesky factorization and the entries h_{ij} of H^T can be computed as follows:*

$$h_{11} = \sqrt{a_{11}}$$

$$h_{ij} = \left(a_{ij} \sum_{k=1}^{j-1} h_{ik} h_{jk} \right) / h_{ij}$$

$$h_{ii} = a_{ii} - \sum_{k=1}^{i-1} h_{ik}^2$$

where the second line is valid for $i = 2, .., n$ and $j = 1, ..., i-1$.

2.5 Iterative Methods ❖

For A nonsingular, suppose we can rewrite the equation $Ax = b$ as $x = Bx + f$, where B is a matrix and f is a vector, so that the solution to the second is the solution to the first[1]. Then using a guessed starting value x_0, we can iteratively determine a sequence $\{x_n\}$ using the relation $x_{n+1} = Bx_n + f$. If the sequence converges to a value $\{x_n\} \to x$, then that value is a solution to $Ax = b$. A few remarks on this method:

- For iterative methods, a sparse A will have a sparse corresponding B. In other words there is no fill-in to the sparse areas of A.
- The sequence $\{x_n\}$ isn't guaranteed to converge, or if it does, it might not converge quickly.

Theorem 2.9. *Given A nonsingular and a consistent iterative method B, f, then $\{x_n\}$ converges to some x for all x_0 if and only if $\rho(A) < 1$, where $\rho(A) = \max_j |\lambda_j(A)|$, where $\lambda_j(A)$ are eigenvalues of A.*

2.5.1 Jacobi and Gauss-Seidel methods

Suppose we wish to find a consistent method B, f when A as no zeros on its diagonal. Then we write $A = D - E - F$, where D is the diagonal of A, E is the lower triangular part, and F is the upper triangular part. Then:

$$Dx = (E + F)x + b \Longrightarrow x = D^{-1}(E + F)x + D^{-1}b$$

Take $B = B_J = D^{-1}(E + F)$ and $f_J = D^{-1}b$. This is called the *Jacobi method*. Component-wise, this takes the form:

$$x^i_{n+1} = \frac{1}{A_{ii}} \left(\sum_{j<i} A_{ij} x^j_n + \sum_{j>i} A_{ij} x^j_n \right) + \frac{1}{A_{ii}} b^i$$

We could have also written:

$$(D - E)x = Fx + b \Longrightarrow x = (D - E)^{-1}Fx + (D - E)^{-1}b$$

Since $(D - E)$ is invertible by assumption. Taking $B = B_{GS} = (D - E)^{-1}F$ and $f_{GS} = (D - E)^{-1}$ gives us what is known as the *Gauss-Seidel method*.

Example 2.10. Suppose A has 2's on its diagonal and -1 on its off-diagonals and $b = 0$. This is a particular type of matrix that appears in differential equations. Then the Jacobi equation is:

$$x^j_{n+1} = \frac{1}{2} \left(x^{j-1}_n + x^{j+1}_n \right)$$

This takes each component and replaces it with the average of its neighboring components. This is therefore a discrete approximation of the heat equation $-u_{xx} = 0$, $u(0) = u(1) = 0$. Starting at $x_0 = (0, 1, 1, ..., 1, 1, 0)$, we get slow monotone convergence $x_n \to 0$.

The slow convergence of the example above is an example of *over-relaxation*. This can be corrected by introducing a parameter $0 < \omega < 1$ and replacing B_J and B_{GS} by:

$$B_{J\omega} = \omega B_J - (1 - \omega)I$$

$$B_{GS\omega} = B(\omega) = (I - \omega D^{-1}E)[(1 - \omega)I + \omega D^{-1}F]$$

These are known as the (over or under)-relaxed Jacobi/Gauss-Seidel matrices.

Exercise 2.11. Show that the standard Jacobi method from Example 2.10 applied to a saw-tooth wave produces another saw-tooth wave. What happens when we use a relaxed Jacobi method, say with $\omega = \frac{1}{2}$?

Lecture 9/13

Lemma 2.12. *If $\rho(B) < 1$, there exists an induced norm such that $\|B\|_* < 1$.*

Proof:

[1] This is called *consistent*.

The idea is to define $\|x\|_* = \|Sx\|_\infty$ for some nonsingular S. Then:

$$\|B\|_* = \sup_{x\neq 0} \frac{\|SBx\|_\infty}{\|Sx\|_\infty} = \sup_{y\neq 0} \frac{\|SBS^{-1}y\|_\infty}{\|y\|_\infty}$$

Therefore we would like to find S such that $\|SBS^{-1}\|_\infty < 1$. There exists a nonsingular matrix T such that $TBT^{-1} = J$ where J is in Jordan normal form. Let $S = DT$, where D is a diagonal matrix D such that $D_{ii} = d^i$ for some d. Then DJD^{-1} will still be a block diagonal matrix and it's diagonals the same as J, but the off-diagonals will be d^{-1}. Therefore $\|DJD^{-1}\|_\infty = \max_i(\lambda_i) + \frac{1}{d} = \rho(B) + \frac{1}{d}$. We can choose d sufficiently large so that this becomes less than 1. Therefore:

$$\|SBS^{-1}\|_\infty = \|DTBT^{-1}D^{-1}\|_\infty = \|DJD^{-1}\|_\infty < 1$$

□

Remark 2.13. If $\rho(B) = 1$ and B has distinct eigenvalues, then $\|B^n\|$ is bounded independent of n by the above construction.

Proof of Theorem 2.9:

Let x be an arbitrary vector. If $\rho(A) \geq 1$, then define $e_{n+1}x_{n+1} - x = Ax_n + f - (Ax + f) = Ae_n$. Therefore $e_n = A^n e_0$. For $e_0 = \tilde{x}$, an eigenvector for $\tilde{\lambda}$ such that $|\tilde{\lambda}| \geq 1$, then $e_n = \tilde{\lambda}]^n e_0 \not\to 0$ and hence x_n does not converge.
If $\rho(A) < 1$, then $\|e_n\|_* \leq \|B\|_*^n \|e_0\|_*$, where the norm is the one from Lemma 2.12. Then $\|e_n\|_* \to 0$ as $n \to \infty$ and hence $e_n \to 0$.

□

Remark 2.14. Compare convergence of general fixed point iteration: $x_{n+1} = g(x_n)$. This converges if $\|\frac{dg}{dx}\| \leq r < 1$, but there is no "only if". So Theorem 2.9 is stronger than generic iteration theorems.

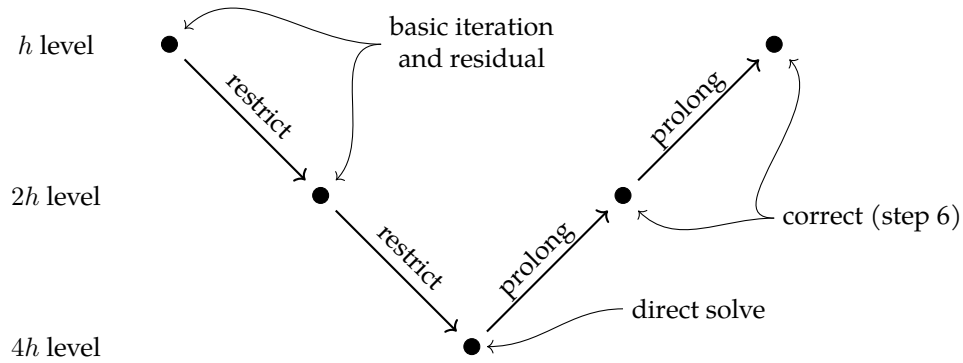
2.5.2 Multigrid Methods

A more practical and more used iterative method is the multigrid method. This comes from approximation of PDE's. Consider the problem $A_h x_h = b_h$, where h is a parameter. We consider x_h as a grid of unknown values, where h is the spacing of the grid. Note that there are redundant grids associated to this system (e.g. $2h, 4h, ...$), hence the name multigrid. Compare this to Example 2.10, where we had a one dimensional grid on which we were solving the heat equation. Coarser grids lead to faster convergence, but have less accuracy. The multigrid method uses the hierarchical grid to balance convergence speed and accuracy.

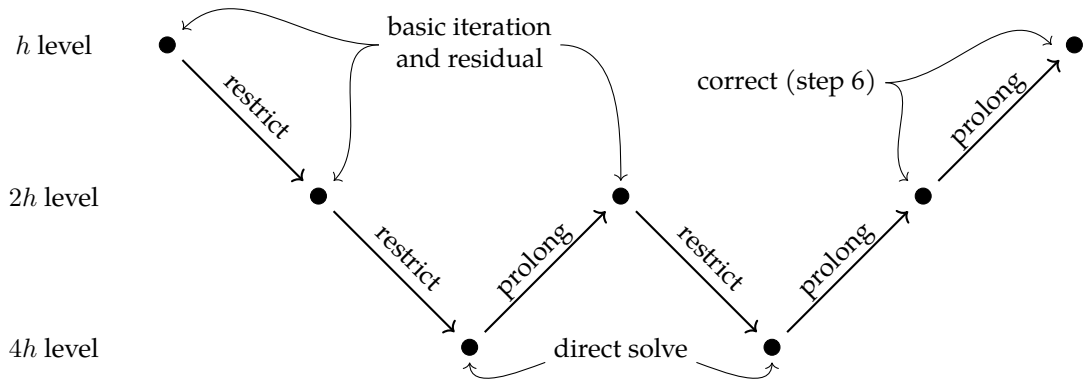
Two-grid algorithm

Define the following algorithm starting from the kth step:

1. Given x_h^k, apply a basic iteration method for a few steps to get $\tilde{x}_h^k$. The preferred method is to use Jacobi iteration (i.e. use B_{J_ω} for $|\omega| < 1$).
2. Calculate the residual $r_h = b_h - A_h\tilde{x}_h$.
3. Restrict to a coarser grid $r_{2h} = I_h^{2h} r_h$, where I_h^{2h} is the interpolation matrix taking r_h to a lower dimension.
4. Solve $A_{2h}y_{2h} = r_{2h}$ on the coarse $2h$ grid, say by Gaussian Elimination.
5. Now we *prolong* the vector y_{2h} to get $y_h = I_{2h}^h y_{2h}$. The matrix I_{2h}^h could, for example, use linear interpolation on the unknown elements.
6. Set $x_h^{k+1} = \tilde{x}_h + y_h$ and return to Step 1.



(a) A V schematic.



(b) A W schematic.

Figure 2.1: Two examples of a multigrid iteration cycle.

More general multigrid methods apply the same ideas with more coarse grids (iteratively on Step 4). See Figure 2.1

Remark 2.15. The calculation performed in Steps 2-6 is summarized as:

$$x_h^{k+1} = \tilde{x}_h + I_{2h}^h A_{2h}^{-1} I_h^{2h}(b_h - A_h \tilde{x}_h)$$

If we set every $2h$ to h, then the interpolation operators are identities and this collapses to $x_h^{k+1} = A_h^{-1} b_h$, which is the solution. If $2h$ and h are close in value, then the interpolation matrices behave "close" to identity operators and hence we would expect x_h^{k+1} to be close to the real solution x_h.

The two-grid method has the form of a basic iteration method with matrix:

$$B_{2G} = (I - I_{2h}^h A_h^{-1} I_h^{2h}) B_{J\omega}^N$$

where N is the number of Jacobi iterations from Step 1 and $B_{J\omega}$ is the under-relaxed Jacobi matrix. By Theorem 2.9, this converges iff $\rho(B_{2G}) < 1$. The convergence rate is faster the smaller $\rho(B_{2G})$ is.

Lecture 9/16

2.5.3 Gradient method

Suppose A is symmetric and positive definite. The gradient method (also known as steepest descent) seeks to minimize any function $f(x)$ using its gradient. We define the recurrence:

$$x^{k+1} = x^k + \alpha_k p^k \tag{†}$$

where α_k is a scalar and $p^k = -\nabla f(x^k)$. The parameter α_k is determined by searching along the line $\mathrm{span}(p^k)$ and determining which value of α_k produces the smallest value of $f(x^{k+1})$. For our purposes, solving $Ax = b$ is equivalent to minimizing the quadratic form:

$$f(y) = \phi(y) = \frac{1}{2} y^T A y - y^T b$$

which is known as the *energy*. Then we have:

$$\nabla \phi = \frac{1}{2}(A^T + A)y - b = Ay - b$$

since we assumed A is symmetric. Indeed, a solution to $\nabla\phi = 0$ is also a solution to the original equation. Conversely, if x is a solution to $Ax = b$, then x minimizes ϕ:

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

This is $\phi(x)$ plus some positive quantity; to minimize $\phi(y)$ we just need that quantity to be zero, which happens when $y = x$. Therefore minimizing ϕ is equivalent to solving $Ax = b$. Since we are minimizing a quadratic form, our gradient descent will have a unique minimum (we are traveling along quadric level sets).

To find α_k for our linear system, we need to choose it so that it minimizes:

$$\phi(x^{k+1}) = \frac{1}{2}(x^k + \alpha_k p^k)^T A(x^k + \alpha_k p^k) - (x^k + \alpha_k p^k)b$$

Setting the derivative $\frac{d\phi}{d(\alpha_k)}$ to zero, we get:

$$\alpha_k = \frac{(p^k)^T r^k}{(p^k)^T A p^k}$$

Notice that the vector p^k is the residual:

$$p^k = -\nabla\phi(x^k) = b - Ax^k = r^k$$

We will keep them distinct for reference in future sections where they are different. Putting it all together, the recursion (†) is:

$$\begin{cases} x^{k+1} = x^k + \alpha_k r^k \\ r^{k+1} = r^k - \alpha_k A r^k \\ \alpha_k = \frac{(p^k)^T r^k}{(p^k)^T A p^k} \\ r^k = p^k \end{cases} \tag{2.5.1}$$

Keep in mind that $r^0 = b - Ax^0$, so only x^0 is required for this recursion to be defined.

Definition 2.16. Given a matrix A and a vector b, the associated *Krylov subspace* is:

$$K_r(b, A) = \mathrm{span}\{b, Ab, A^2 b, A^3 b, ..., A^{r-1} b\}.$$

A Krylov subspace method consists of finding an approximate solution to the problem $Ax = b$ that lives in $K_r(\tilde{b}, A)$ for some $\tilde{b}$. The motivation for this subspace is computational efficiency – the subspace can be easily determined by matrix-vector multiplication. The Krylov method can also be used to solve eigenvalue problems. The example of gradient descent we just did is an example of a Krylov subspace method because we only have

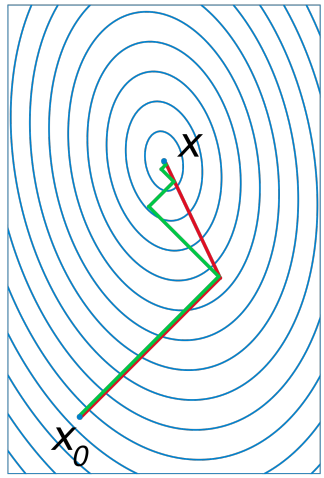


Figure 2.2: Convergence comparison of gradient descent (green) with conjugate gradient (red). Source: Wikipedia

to compute vectors in the space $K_1(A, r^k)$.

The conjugate gradient method is similar, but uses A-orthogonal search directions. This will then imply $(p^k)^T A p^{k+1} = 0$. This can often lead to faster convergence because it avoids zig-zagging to the minimum (see Figure 2.2). The recursion formulae are:

$$\begin{cases} x^{k+1} = x^k + \alpha_k p^k \\ r^{k+1} = r^k - \alpha_k A p^k \\ p^{k+1} = r^{k+1} - \beta^k p^k \\ \beta^k = \frac{(Ap^k)^T r^{k+1}}{(Ap^k)^T p^k} \end{cases} \tag{2.5.2}$$

where α_k is as before. See [Qa] §4.3.4 for a detailed derivation. Note that this is also a Krylov subspace method. More importantly, it is actually secretly a direct method. This follows from the fact that $\{r^0, ..., r^{k-1}\}$ forms an orthogonal basis of $K_k(r^0, A)$, and hence $r^j = 0$ for some $j \leq n$ by linear dependence. Therefore $e^j = 0$ eventually and this process terminates after at most n steps. The conjugate gradient method after k steps gives the best approximation to the solution in $K_k(r^0, A)$. That is:

$$\|x - x^k\|_2 = \min_{y \in K_k(r^0, A)} \|x - x^k - y\|_A$$

Remark 2.17. Our initial requirement was that $\langle p^k, p^{k+1} \rangle_A = 0$. However, it turns out that $\langle p^i, p^j \rangle_A = 0$ for all $i \neq j$. The vectors p^i are known as the *conjugate directions*. For more discussion, see [Qa] §4.4.

Exercise 2.18. Apply the conjugate gradient method to $Ax = b$, where

$$A_{i,j} = \begin{cases} 2 & i = j \\ -1 & |i - j| = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$b_j = 1$$

and $1 \leq i, j \leq 50$. Start with $x^0 = 0$ and $p^0 = b - Ax^0 = b$.

Lecture 9/30

2.5.4 Error in Gradient Methods

Notice the gradient descent formulae can be expressed as:

$$x^{k+1} = (I - \alpha_k A)x^k + \alpha_k b$$

Which is nearly the form of an iterative equation, except B and f are no longer constant as in Gauss-Seidel or Jacobi. Using this, we can calculate the error:

$$\epsilon^{k+1} := x^{k+1} - x = (I - \alpha_k A)(x^k - x) = (I - \alpha_k A)\epsilon^k$$

Notice that:

$$\|\epsilon^{k+1}\|_2 \leq \min_\alpha \|I - \alpha A\|_2 \|\epsilon^k\|_2$$

We would like the error to converge to 0, which would require $\min_\alpha \|I - \alpha A\|_2 < 1$. Since A is symmetric, we can diagonalize it to a diagonal matrix Λ. Since this can be done via an orthogonal transformation, we have:

$$\begin{aligned}\min_\alpha \|I - \alpha A\|_2 &= \min_\alpha \|I - \alpha \Lambda\|_2 \\ &= \min_\alpha (\max(\alpha\lambda_M - 1, 1 - \alpha\lambda_m))\end{aligned}$$

where λ_M and λ_m are the largest and smallest eigenvalues of A, respectively. We would like to choose α so that the maximum is achieved, i.e. such that $\alpha\lambda_M - 1 = 1 - \alpha\lambda_m$. Thus $\alpha = \frac{2}{\lambda_M + \lambda_m}$. After some algebra, we find:

$$\min_\alpha \|I - \alpha A\|_2 = \frac{\frac{\lambda_M}{\lambda_m} - 1}{\frac{\lambda_M}{\lambda_m} - 1}$$

but $\|A\|_2 = \lambda_M$ and $\|A^{-1}\|_2 = \frac{1}{\lambda_m}$. Therefore $\frac{\lambda_M}{\lambda_m}$ is the condition number $K_2(A)$. Finally, for (2.5.1) to converge, we need:

$$\frac{K_2(A) - 1}{K_2(A) + 1} < 1$$

We can do a similar calculation for the error in the conjugate gradient method:

$$\|\epsilon^k\|_A \leq 2\left(\frac{\sqrt{K_2(A)} - 1}{\sqrt{K_2(A)} + 1}\right)^k \|\epsilon^0\|_A$$

We see a similar term to the gradient method, except with square roots on the condition number $K_2(A)$. This can be a significant advantage for certain values of $K_2(A)$.

2.5.5 Pre-conditioning

For both of these gradient methods, $K_2(A) \gg 1$ means the convergence will be slow in general. One pre-process step to improve this is to reduce K_2 by *pre-conditioning*. Notice that for any non-singular B, we can transform:

$$Ax = b \Rightarrow BAx = Bb$$

Pre-conditioning consists of finding a matrix B such that $K_2(BA)$ is small, so that we can instead perform gradient methods on the transformed equation. Of course, we could choose $B = A^{-1}$, but this is expensive and defeats the purpose. So we need to compute B cheaply (one can think of it as a cheaper pseudo-inverse to A). A few ways to do this are:

- Perform a few multigrid steps to get an approximation to $B = A^{-1}$.
- Do an "incomplete" LU decomposition of A, which is only an approximation to the LU decomposition.

2.6 Eigenvalue problems ❖

The eigenvalue problem consists of finding a nonzero λ such that $Ax = \lambda x$ for $x \neq 0$. The methods that we use to approach this problem fall into one of the following classes:

- Power method, which considers the sequence $\{Ax, A^2x, A^3x, ..., \}$.
- Using determinants.
- Similarity transformations, e.g. QR.
- Krylov subspace methods.
- Special matrix methods

Lecture 10/2

2.6.1 Power method

Let the eigenvalues of A be ordered such that $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ....$ Note we are assuming there is a unique greatest eigenvalue λ_1. If A is diagonalizable, we can expand any x in an eigenbasis $\{x_j\}$ and we find:

$$A^k x = \sum_{j=1}^{n} \alpha_j \lambda_j^k x_j = \alpha_1 \lambda_1^k \left(x_1 + \sum_{j=2}^{n} \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right)$$

Since λ_1 is largest, for $k \to \infty$ we have $A^k x \to \alpha_1 \lambda_1^k x_1$. The theorem below formulates this more precisely.

Theorem 2.19. *Let A be diagonalizable having a unique largest eigenvalue λ_1 with a corresponding eigenvector x. Then there exists $c > 0$ such that:*

$$\|q^{(k)} - x\|_2 \leq c \left(\frac{\lambda_2}{\lambda_1} \right)^k$$

where q is defined by the recursion:

$$q^{(k)} = \frac{z^{(k)}}{\|z^{(k)}\|_2}$$
$$z^{(k)} = Aq^{(k-1)}$$

for any starting value of $q^{(0)}$. Moreover, as $k \to \infty$:

$$\nu^{(k)} = (q^{(k)})^\dagger A q^{(k)} \to \lambda_1$$

Remark 2.20. We can use the same sequence as above to approximate the smallest eigenvalue (assuming it is nonzero) by applying it to A^{-1}. In fact, we can pick out any eigenvalue by applying it to $(A - \mu I)^{-1}$, where μ is any complex number. Doing the power method here will converge the eigenvalue closest to μ, since the eigenvalues of $(A - \mu I)^{-1}$ are $(\lambda_j - \mu)^{-1}$ and the largest one is the one closest to μ.

2.6.2 QR Algorithm

The QR method uses similarity transformations to bring A into triangular form. The advantage is that it allows us to read off the entire spectrum of A. Consider the recursion:

$$\begin{cases} Q^{(k)} R^{(k)} = T^{(k-1)} \\ T^{(k)} = R^{(k)} Q^{(k)} \end{cases}$$

The first line is a QR decomposition. This process is initialized as $T^{(0)} = (Q^{(0)})^T A Q^{(0)}$ so that $T^{(k)}$ is almost diagonal (made precise next).

Theorem 2.21. *Suppose the eigenvalues $\{\lambda_j\}$ of A are all distinct. Then:*

$$\lim_{k\to\infty} T^{(k)} = \begin{pmatrix} \lambda_1 & t_{1,2} & \cdots \\ 0 & \ddots & t_{n-1,n} \\ 0 & 0 & \lambda_n \end{pmatrix}$$

$$|t_{i,i-1}^{(k)}| = \mathcal{O}\left(\left(\frac{|\lambda_i|}{|\lambda_{i+1}|}\right)^k\right)$$

If A is symmetric, then $T^{(k)}$ converges to a diagonal matrix.

Exercise 2.22. Partially prove the Theorem by showing that the spectrum of (the limit of) $T^{(k)}$ is the same as the spectrum of A, assuming it converges.

For computational stability and speed, it is common to use *Householder transformations* to get $Q^{(0)}$ and *Givens rotations* for the QR step. These are orthogonal transformations that can be used to introduce zeros in matrix-vector multiplication.

Definition 2.23. Let $P = I - 2vv^T/\|v\|_2^2$, where $v = x \pm \|x\|_2 e_m$. Then P is called the Householder transformation associated to x.

The property of the Householder transformation P associated to x is that $Px = (0, ..., 0, \|x\|_2, 0, ..., 0)$ (where the mth component is nonzero). One can check that P is orthogonal and symmetric. These transformations are used to get $Q^{(0)}$, which make $T^{(0)}$ into *upper Hessenberg* form. This is a matrix whose entries below the subdiagonal are zero. This is done by taking:

$$(Q^{(0)})^T = \begin{pmatrix} I_{n-1} & \\ & P_{n-1} \end{pmatrix} \cdots \begin{pmatrix} I_2 & \\ & P_2 \end{pmatrix} \begin{pmatrix} I_1 & \\ & P_1 \end{pmatrix}$$

where P_k are appropriate Householder matrices (of size $(n-k) \times (n-k)$) and I_k is the $k \times k$ identity matrix. Now assuming $T^{(k-1)}$ is upper Hessenberg, the next step is to perform a QR decomposition to get $Q^{(k)}$ and $R^{(k)}$. In other words, we want to find $Q^{(k)}$ such that $(Q^{(k)})^T T^{(k-1)}$ is upper-triangular. Typically, Givens rotations are used here (though Householder can be used as well).

Definition 2.24. For an angle θ and indices i, k, the Givens rotation is the matrix:

$$G(i,k,\theta) = \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & \cos\theta & \dots & \sin\theta & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -\sin\theta & \dots & \cos\theta & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix}$$

This is the matrix with ones on the diagonal, except at the ith and kth index, where the diagonal is $\cos\theta$. The only other nonzero entries are $\sin\theta$ and $-\sin\theta$ at the (i,k) and (k,i) positions.

A Givens rotation can be used to introduce a zero component in a vector by picking θ such that $\tan\theta = x_i/x_k$. For example:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ 0 \end{pmatrix}$$

Since $T^{(k-1)}$ is upper Hessenberg, we can apply a sequence of Givens rotations to remove the nonzero subdiagonals to get a triangular matrix. This sequence of Givens matrices is our desired $Q^{(k)}$.

Lecture 10/7

We can generalize Theorem 2.21 when A is real. In this case, the eigenvalues are in conjugate pairs but can be arbitrary otherwise. The matrices $T^{(k)}$ are almost upper triangular, except the diagonal is of 2×2 matrices. Using the determinant, we can extract the eigenvalue pairs from each diagonal 2×2 matrix.

While the QR method is the modern standard for computing spectra, there are other methods worth mentioning like the Jacobi method, which uses Givens rotations, Sturm sequences, and Krylov subspace methods for sparse matrices.

2.6.3 Eigenvectors

The QR method gives us a similarity transformation $Q^T AQ = T$ for T upper triangular and Q orthogonal. f $Ax = \lambda x$, then $QAQQ^T x = \lambda Q^T x$, and hence $Ty = \lambda y$ where $y = Q^T x$. Thus the eigenvector x can be recovered by solving $Ty = \lambda y$ and then computing $x = Qy$. The equation $Ty = \lambda y$ is of the block form:

$$\begin{pmatrix} T_{11} & v & T_{13} \\ 0 & \lambda & w^T \\ 0 & 0 & T_{33} \end{pmatrix} \begin{pmatrix} y_{k-1} \\ \alpha \\ y_{n-k} \end{pmatrix} = \lambda \begin{pmatrix} y_{k-1} \\ \alpha \\ y_{n-k} \end{pmatrix}$$

This breaks into the equations:

$$\begin{aligned} (T_{11} - \lambda I_{k-1})y_{k-1} + \alpha v + T_{13} y_{n-k} &= 0 \\ w^T y_{n-k} &= 0 \\ (T_{33} - \lambda I_{n-k})y_{n-k} &= 0 \end{aligned}$$

Solving this for $y = [y_{k-1}, \alpha, y_{n-k}]$ gives:

$$y = \begin{pmatrix} -(T_{11} - \lambda I_{k-1})^{-1} v \\ 1 \\ 0 \end{pmatrix}$$

2.6.4 Estimating Eigenvalues

Theorem 2.25 (Gershgorin)**.** *Let A be a complex $n \times n$ matrix and for every i define:*

$$\mathcal{R}_i = \left\{ z \in \mathbb{C} : \ |z - a_{ii}| \leq \sum_{j=1, j \neq k}^{n} |a_{ij}| \right\}$$

Then:

$$\sigma(A) \subset S_{\mathcal{R}} := \bigcup_i \mathcal{R}_i$$

The sets $\mathcal{R}_i$ are called *Gershgorin* circles. This theorem says that the spectrum of A lies in a union of circles whose centers and radii are determined by the entries of A.

Proof:

Write $A = D + E$, where D is diagonal and E is off-diagonal. For $\lambda \in \sigma(A)$, we can assume $\lambda \neq a_{ii}$. Define $B_\lambda = A - \lambda I = (D - \lambda I) + E$, which is singular. Then there exists $x \neq 0$ such that $B_\lambda x = 0$, which implies $((D - \lambda I) + E)x = 0$. Therefore:

$$x = -(D - \lambda I)^{-1} E x$$

Taking the norm, we have:

$$\begin{aligned} \|x\|_\infty &\leq \|(D - \lambda I)^{-1} E\|_\infty \cdot \|x\|_\infty \\ \implies 1 &\leq \|(D - \lambda I)^{-1} E\|_\infty \end{aligned}$$

By definition of the infinity norm on matrices, this becomes:

$$1 \leq \sum_{j=1}^{n} \frac{|e_{kj}|}{|a_{kk} - \lambda|} = \sum_{j=1, j\neq k}^{n} \frac{|a_{kj}|}{|a_{kk} - \lambda|}$$

Dividing both sides by $1/|a_{kk} - \lambda|$ proves the result.

□

2.7 Singular Value Decomposition ❖

A singular value decomposition of a matrix A is a decomposition of the form $U^T AV = \Sigma$, where Σ is diagonal and U, V are orthogonal. This can also be done for non-square matrices, where instead $U^T AV = \binom{\Sigma}{0}$ or $(\Sigma \;\; 0)$. The singular value decomposition can be computed efficiently using a variation of the above described QR method. See [Qa] for details.

Lecture 10/9

2.7.1 Over and under determined systems

An over determined system is $Ax = b$ for A an m by n matrix, $m > n$. Typically the solution to this is defined to be $\operatorname{argmin}_x \|Ax - b\|_2$, since no solution is guaranteed for an overdetermined system. The SVD can be used as follows:

$$\min_x \|Ax - b\|_2 = \min_x \|U^T AVV^T x - U^T b\|_2 = \min_y \left\| \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} y - \tilde{b} \right\|_2$$

where we set $y = V^T x$ and $\tilde{b} = U^T b$. Then the solution is $y = \Sigma^{-1}\tilde{b}$, which is the least squares solution. Note that this assumes that Σ has no non-zero diagonals. If it had zeros on diagonals, WLOG let them be in the last $n - p$ rows. Let Σ_p be the nonsingular diagonal submatrix of Σ consisting of the first p rows. Then we set $y' = \Sigma_p^{-1}\tilde{b}'$ and our solution is $[y' \;\; y'']$, where y'' is arbitrary. Since we have freedom in choosing y'', the solution is not unique. Often we choose $y'' = 0$ to get the minimal length vector.

The under determined system is $Ax = b$ for A as above, but $m < n$. In this case, there is typically not a unique solution. If we decompose A using SVD as before assuming the resulting Σ is invertible, then (using notation above) any solution is of the form $[y_1 \;\; y_2]$, where $y_1 = \Sigma^{-1}b$ and y_2 is arbitrary (typically chosen to be zero). Once again, if Σ is singular, then we just do the same thing on Σ_p and our solution is $[y' \;\; y'' \;\; y_2]$, where $y' = \Sigma_p^{-1}b$, and y'', y_2 are arbitrary.

These processes can be summarized by defining the Morse-Penrose inverse:

Definition 2.26. For a matrix A with SVD $A = U\Sigma V^T$, with $\sigma_1, ..., \sigma_p$ the nonzero diagonals of Σ (presumed to be in the first p rows), then define $\Sigma^+ = \operatorname{diag}(\sigma_1^{-1}, ..., \sigma_p^{-1}, 0, ..., 0)$. The *Morse-Penrose inverse* is $A^+ = U\Sigma^+ V^T$.

Therefore the solution to an under determined or over determined system is $x = A^+ b$.

2.7.2 Application to matrix compression

The compression of low rank matrices is a computational use of SVD. Suppose A has low rank and compute the SVD $A = U\Sigma V^T$. The large number of zeros in Σ (including many diagonals) means that we can do away with many parts of U and V in the matrix multiplication. That is, $A = \mathcal{U}\Sigma_p \mathcal{V}^T$, where $\mathcal{U}, \mathcal{V}$ are much "thinner" than U and V and Σ_p is the nonsingular part of Σ. This has the potential to speed up vector multiplication for A when its rank is very low when compared to the size of A.

2.7.3 Fast Multipole Method

Here we provide a brief example that describes what the Fast Multipole Method is. Let $h > 0$ and let A be the matrix:

$$A = \begin{cases} a_{ij} = \frac{1}{h|i-j|} & i \neq j \\ a_{ij} = 0 & i = j \end{cases}$$

We see that for (i, j) very far from the diagonal, the entry a_{ij} is small. For concreteness, let A be 10,000 by 10,000 and $h = 0.01$. Then to perform the matrix multiplication Ax efficiently, we divide A into 100 by 100 blocks and precompute the sums $x_1 + ... + x_{100}, x_{101} + ... + x_{200}$, etc. Then we approximate each 100 by 100 sub-matrix multiplication using the midpoint of each row and the precomputed sums, since the rows are approximately constant. For example, the ith row of the top right block of A multiplied by the last 100 entries of x is approximated as:

$$\sum_{j=10000-100}^{10000} a_{ij}x_j \approx a_{iJ}(x_{J-50} + ... + x_{J+50})$$

where $J = 10000 - 50$. We should only use this approximation for the blocks in A that are far from the diagonal. For blocks closer, we use standard direct sum computations.

There are some improvements that can be made to the method we just described. One is to choose different sized subdivisions of A, creating a hierarchical decomposition of A. Another one is to use a better-than-constant approximation to the rows of A (linear, quadratic, etc). This is more costly, but the resulting approximation to A is much better.

Example 2.27. As an analogy/application, consider the n-body problem. Each time step, we need to compute the force on each of the n objects from the other $n - 1$ objects, which results in $\mathcal{O}(n^2)$ work each step. To speed this up, we can clump together body clusters that are far away and approximate them as a point whose mass is the sum of their masses (precomputed).

3. Nonlinear Equations

❖

Lecture 10/14

Suppose $f : \mathbb{R}^d \to \mathbb{R}^d$ is nonlinear and continuous (possibly differentiable). The standard ways that the equation $f(x) = 0$ is solved numerically are using iterative methods and algorithms based on the evaluation of $f(x)$ and $\nabla f(x)$.

For $d = 1$, a simple example of a method which only uses the evaluation of f is the bisection method. This uses the intermediate value theorem. This consists of finding two points a, b such that $f(a) < 0$ and $f(b) > 0$. Then we check the endpoints of the intervals $[a, m]$ and $[m, b]$, where m is the midpoint, and repeat on whichever interval has different sign values of f on the endpoints. The error estimate for this method after n steps is:

$$|m - \bar{x}| = \frac{b-a}{2^n}$$

where $\bar{x}$ is the solution. Unfortunately, this algorithm doesn't generalize to higher d; however, there are similar search methods in higher dimensions for finding minima.

3.1 Fixed point iteration ❖

Suppose we can find a way to rewrite the system $f(x) = 0$ as $g(x) = x$ *consistently* (i.e. they have the same solution set). We can compare this to the iterative methods for linear equations. Just as with those, we can define a sequence $x_j = g(x_{j-1})$ for a given x_0. If this converges, then the limit is a solution to our initial equation. There is also a way to characterize the convergence based on the gradient of g. Let x be a solution; then:

$$\|x_{j+1} - x\| = \|g(x_j) - x\| = \|g(x + (x_j - x)) - x\| \leq \|(\nabla g)_x\| \cdot \|x_j - x\| \tag{3.1.1}$$

where we Taylor expanded $g(x + (x_j - x))$ to get the inequality. Therefore if we assume $\|(\nabla g)_x\| \leq r < 1$, then $\|x_j - x\| \leq r^j \|x_0 - x\|$ and we get convergence.

3.1.1 Newton's Method

Newton's method is a particular type of fixed point iteration where we choose g to be $x - Df(x)^{-1}f(x)$, where Df is the Jacobian of f. Then our sequence is:

$$\begin{cases} x_{j+1} = x_j - Df(x_j)^{-1}f(x_j) \\ x_0 \text{ given} \end{cases}$$

To derive this for $d = 1$, note that our g is always of the form $x + a(x)f(x)$ where $a(x) \neq 0$. We wish to pick a such that the derivative of g is smallest (to get convergence).

$$g'(x) = 1 + a'f + af'$$

At a solution $f(x) = 0$, so this reduces to minimizing $1 + a(x)f'(x)$. Clearly $a(x) = -\frac{1}{f'(x)}$ makes this expression zero. The same derivation can be done for higher dimensions too.

Newton's method is quadratically convergent, provided f is C^1 and Df is nonsingular, meaning:

$$\|\bar{x} - x_j\| \leq C\|\bar{x} - x_j\|^2$$

Remark 3.1. This error estimate is an improvement on the generic error estimate 3.1.1. Since we chose g such that $g' = 0$, we have $r = 0$ and so the bound moves to the quadratic term. Unfortunately, we don't have control over the convergence constant C.

It should also be noted that, like all nonlinear iterative methods, there is no guarantee that Newton's method will converge for any starting point. Moreover, calculating Df^{-1} numerically is often costly. For this reason, a preferred fixed point method is the next two methods, which approximate the derivative using secants.

3.2 Secant Method & Quasi-Newton method ❖

The secant method for $d = 1$ allows us to bypass evaluating the derivative by approximating it as a secant. The recursion is:

$$x_{j+1} = x_j - \frac{x_j - x_{j-1}}{f(x_j) - f(x_{j-1})} f(x_j)$$

Lecture 10/21

For larger d, there is the Quasi-Newton method that can be seen as a generalization of the secant method. For each k, define the matrix Q_k such that:

$$Q_k \delta x_k = b_k \qquad \text{(Approx. to Jacobian)}$$

where $\delta x_k = x_{k+1} - x_k$ and $b_k = f(x_k) - f(x_{k+1})$. Then we use the recursion:

$$x_{k+1} = x_k - Q_k^{-1} f(x_k)$$

The above approximation to the Jacobian defines n equations with n^2 unknowns (the entries of Q_k). There are therefore many choices of Q_k and hence different strategies.

One strategy is to use Q_{k-1} to compute Q_k, which saves storage. This is known as the *Broyden* method:

$$Q_k = Q_{k-1} + \frac{(b_k - Q_{k-1}\delta x_k)(\delta x_k)^T}{(\delta x_k)^T \delta x^k}$$

This is obtained by minimizing the Frobenius norm $\|Q_k - Q_{k-1}\|_F$. It can be used under the assumption that $\|x_0 - x^*\| \leq \epsilon$ and $\|Q_0 - \partial f / \partial x(x^*)\| \leq \gamma$, where x^* is a solution and ϵ, γ are sufficiently small.

3.2.1 Special Methods for polynomials

If $f(x)$ is a polynomial, we can factor it by finding any root $x = z$, and dividing by $(x - z)$ to get $g(x)$ such that $f(x) = (x - z)g(x)$. Then we repeat for $g(x)$ until its degree is 1. Typically Newton's method is used for root finding and Horner's method is used to quicken evaluations of f and f'. It should be noted that, if there are complex roots, we could use a $d = 2$ method on $f = (\Re(f), \Im(f))$ or a $d = 1$ method carrying out all arithmetic over $\mathbb{C}$.

3.2.2 Stopping Criteria

For all of the above methods, there are two reasonable stopping criteria:

$$\|f(x_k)\| \leq \epsilon \tag{3}$$

$$\|x_k - x_{k-1}\| \leq \epsilon \tag{4}$$

where ϵ is some chosen tolerance level. In either case, we would like the accuracy $\|x_k - x^*\|$ to be as small as possible. Depending on the method used, we might use one of these over the other for higher accuracy.

If we use condition (3), the size of $\|\frac{\partial f}{\partial x}\|$ determines how close x_k is to the real solution; if it is large, then x_k is closer and if it is small then x_k is further. If instead we use condition (4), the accuracy depends on the convergence rate of the particular method you use.

4. Optimization

❖

Lecture 10/23

Suppose $f : \mathbb{R}^d \to \mathbb{R}$. The unconstrained minimization problem consists of finding $\operatorname{argmin}_{x\in\mathbb{R}^d} f(x)$. If there is some constraint $x \in \Omega \subset \mathbb{R}^d$, then finding $\operatorname{argmin}_{x\in\Omega} f(x)$ is the constrained minimization problem.

4.1 Unconstrained Minimization ❖

For $d = 1$, there is a search algorithm similar to the bisection method. Suppose f is continuous and there exist points x_1, x_2, x_3 such that $f(x_2) < \min(f(x_1), f(x_3))$, then there is a minimum in $[x_1, x_3]$. Then we pick $x_4 \in [x_1, x_2]$ and $x_5 \in [x_2, x_3]$. Then among $x_1, ..., x_5$, we choose three points x_i, x_j, x_k such that $f(x_j) < \min(f(x_j), f(x_k))$ and repeat (note we shouldn't take $i = 1, j = 2, k = 3$, as that would bring us back to the initial triple). Eventually this will converge to a local minimum.

4.1.1 The Nelder and Mead Simplex Method

For higher d, there are still search algorithms for minimization. The primary one is the simplex method by Nelder and Mead. The idea is to choose a simplex in the domain and evaluate f at the nodes. For example, if $d = 2$, we would choose three points and evaluate f at those points. Then find the point $x^{(M)}$ where f is largest and reflect that node across the sub-simplex spanned by the remaining points to a new point $x^{(R)}$ (see Figure **??**).

How we proceed depends on the value of $f(x^{(R)})$.

1. If $f(x^{(R)}) < f(x^{(M)})$ but not the minimum of all the points, then we replace $x^{(M)}$ by $x^{(R)}$ and repeat on the new simplex.

2. If $f(x^{(R)})$ is smaller than all other values, then we evaluate at $x^{(R')} = x^{(M)} + \beta(x^{(R)} - x^{(M)})$ for some $\beta > 1$. If this value is smaller than $f(x^{(R)})$, then we replace $x^{(M)}$ by $x^{(R')}$ and repeat on the new simplex. Otherwise, we replace $x^{(M)}$ by $x^{(R)}$ as in case 1.

3. If neither of the above cases occur, try a new point $x^{(R'')} = x^{(M)} + \gamma(x^{(R)} - x^{(M)})$ for $\gamma < 1$. If $f(x^{(R'')}) < f(x^{(M)})$, then we take $x^{(R'')}$ to be our new vertex and repeat on the new simplex. If this doesn't hold, then we shrink the simplex by a factor of two, keeping the minimal vertex fixed.

Typically, the values of $\beta = 2$ and $\gamma = \frac{1}{2}$ are chosen.

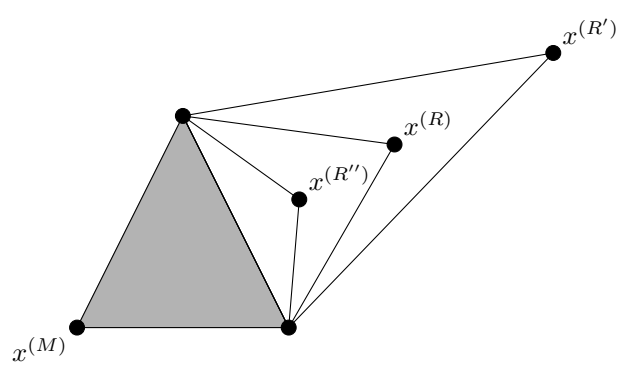


Figure 4.1: A Nelder-Mead simplex with three possible subsequent simplices.

4.1.2 Gradient & Hessian Methods

There are also gradient based methods similar to those described in §2.5.4. Recall that the two methods were gradient descent and conjugate gradient. The former as described by the recursion:

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k)$$

$$\alpha_k = \min_{\alpha} f(x^k - \alpha \nabla f(x^k))$$

where α_k is the minimum of f along the negative gradient direction. This can be found e.g. by $d = 1$ Newton's method or the $d = 1$ search algorithm described at the beginning of this section. If there are many local minima along the line, then we can just fix the step size $\alpha_k = \overline{\alpha}$ for some satisfactory $\overline{\alpha}$.[2]

Now suppose we have more information, namely that $f \in C^2(\mathbb{R}^d)$, giving us the Hessian $Hf = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)$. A sufficient condition for a local minimum $x \in \mathbb{R}^d$ is that $\nabla f(x) = 0$ and $Hf(x)$ is strictly positive definite. We can use Newton's method to find a solution to the system $\nabla f(x) = 0$, so:

$$x^{k+1} = x^k - Hf(x^k)^{-1} \nabla f(x^k)$$

since Hf is the Jacobian of the gradient. The usual Newton's method convergence analysis applies. Another approach is to use Quasi-Newton methods, such as Broyden. The advantage here is that we don't have to compute the Hessian directly each step. The recursion is:

$$\begin{cases} x^{k+1} = x^k - B_k^{-1} \nabla f(x^k) \\ B_{k+1} = B_k + \frac{(y^k - B_k s^k) c^T}{c^T s^k} \end{cases}$$

where $y^k = \nabla f(x^{k+1}) - \nabla f(x^k)$, $s^k = -B_k^{-1} \nabla f(x^k) = x_{k+1} - x_k$, and $c = y^k - B_k s^k$. If we had chosen $c = s^k$, this would have been exactly the Broyden method; but choosing $c = y^k - B_k s^k$ provides a slight optimization.

Lecture 10/25

4.2 Constrained Optimization ❖

If we now wish to minimize $f(x)$ subject to some constraint $x \in \Omega \subset \mathbb{R}^d$, one possible way forward is to reformulate the constraint optimization as unconstrained:

$$\min_{x \in \Omega} f(x) \to \min_{x \in \mathbb{R}^n} g_\epsilon(x)$$

where ϵ is a parameter that is typically present. One possibility for g_ϵ is called the *exterior penalty*:

$$g_\epsilon(x) = f(x) + \epsilon^{-1} d(x, \Omega)^2 \qquad \text{(exterior penalty)}$$

This penalizes x for being outside of Ω inversely proportional to ϵ. As $\epsilon \to 0$, the minimum will approach the minimum in Ω. An alternative to the exterior penalty is the *interior penalty* is:

$$g_\epsilon(x) = f(x) + \epsilon d(x, \Omega^c)^{-1} \qquad \text{(interior penalty)}$$

For both methods, typically one starts with larger ϵ for better regularity and then it is decreased for accuracy.

Remark 4.1. For the exterior penalty choice, the distance $d(x, \Omega)$ is squared to ensure that g_ϵ has C^1 derivative. This comes at the cost of possibly finding exact solutions, however.

Example 4.2. For $d = 1$, suppose we wish to minimize $f(x) = x$ for $\Omega = \{x \in \mathbb{R} \mid 1 \leq x \leq 2\}$. Then the interior and exterior penalty graphs of $g_\epsilon(x)$ for various ϵ are shown in Figure 4.2.

[2] In the context of machine learning, this is known as a *fixed learning rate*.

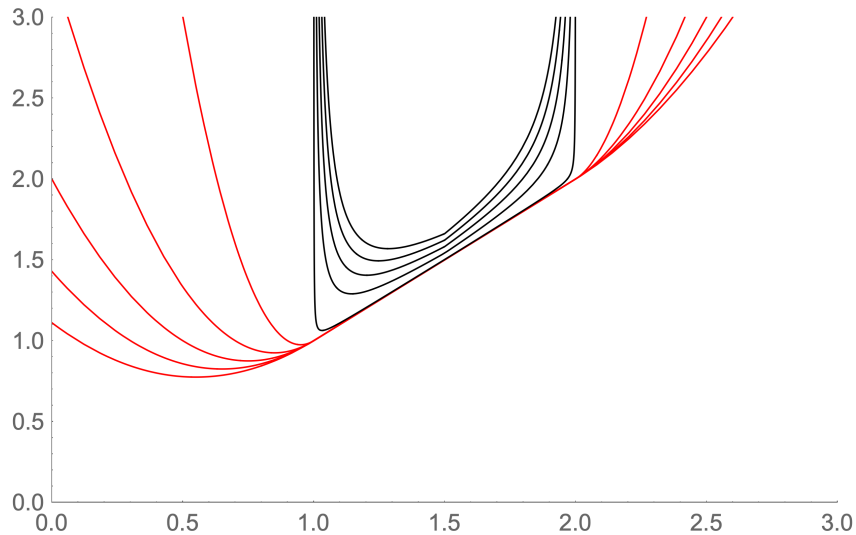


Figure 4.2: The exterior (black) and internal (red) penalty functions $g_\epsilon(x)$ as $\epsilon \to 0$. The minimum of every curve approaches the true minimum of $x = 1$ on $\Omega = [1, 2]$.

4.3 Linear Programming ❖

To solve constrained problems directly, we start with the case where f is linear and Ω is a polytope. Then the constrained optimization problem is called a linear program. The function f is called the objective function and Ω is defines the set linear constraints. The problem can be expressed as:

$$\max c^T x \text{ such that } Ax \leq b, x \geq 0 \tag{4.3.1}$$

where A is a matrix of appropriate dimensions.

Example 4.3. Suppose we wish to minimize $x_1 + x_2$ (e.g. $c = (1, 1)$) such that $x_1 \leq 0, x_2 \leq 1$ and $x_1, x_2 \geq 0$. The polytope Ω is the unit square and the optimum is found at the top right corner.

Remark 4.4. The following are a few important facts that happen with linear programs:

- ◇ The optimum value is never in the interior of Ω, by continuity. An optimum value will always be on a corner (though possibly also on an entire face).
- ◇ Ω could be empty, in which case we say there is no *feasible solution*.
- ◇ Ω could be unbounded and possibly not give a finite solution.
- ◇ The optimum value might not be unique.

The standard solution algorithm is the simplex method by Danzig.[3] The idea is to traverse the polytope corner to corner. The broad overview is:

1. Find a starting corner point on Ω (called a *feasible point*). This is done by choosing a nonsingular maximal set of constraints and checking that the corresponding intersection of hyperplanes (which is a point) lies in Ω.
2. Change one of the constraints defining the current feasible point to another constraint, giving a new feasible point. This is called the pivot step.

[3] This has the same name as the Nelder and Mead simplex method, but is not related.

3. Continue to pivot until the optimum is found.

The pivot step chooses the constraint that is the "best direction" in which to move to get to the solution. We relegate the details of the pivot step to [Pa].

Remark 4.5. The most common alternative to the simplex method is the interior point method. This has a better worst case convergence than the simplex method. It performs better in cases where the simplex method has to visit many or all of the vertices of Ω. However, this is rare for most linear programs.

4.3.1 Bringing LP into canonical form

A linear program in the form of (4.3.1) is in what is called *canonical form*. There are some basic ways to convert an arbitrary linear program to one in canonical form. First, c can always be chosen so that the program is a maximization (by changing the sign of c). Similarly, any constraint $a^T x \geq b$ can be converted to $-a^T x \leq -b$. If there is no constraint on the sign of x_j, then we can replace it with $x_j^1 - x_j^2$, where x_j^1, x_j^2 are two new variables such that $x_j^1, x_j^2 \geq 0$. If a constraint is of the form $a^T x = b$, then we can introduce a new variable x_{n+1} and write $a^T x - x_{n+1} \leq b$ with additional constraint $x_{n+1} \geq 0$.

4.4 Nonlinear Constrained Optimization ❖

The more general case of constrained optimization will not necessarily have linear constraints or objective functions. We consider problems of the form:

$$\begin{array}{lcc} \min f(x) & & f : \mathbb{R}^n \to \mathbb{R} \\ g(x) \leq 0, & \text{where} & g : \mathbb{R}^n \to \mathbb{R}^J \\ h(x) = 0 & & h : \mathbb{R}^n \to \mathbb{R}^K \end{array} \tag{4.4.1}$$

Recall in the unconstrained case, the condition for a minimum is $\nabla f(x) = 0$. With constraints, the conditions for a minimum are called the Kuhn-Tucker conditions. For $\lambda \in \mathbb{R}^J$ and $\mu \in \mathbb{R}^K$, define the Lagrangian:

$$L(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$$

The Kuhn-Tucker conditions are that there exist x_0, λ_0, μ_0 such that:

$$\begin{array}{c} \nabla_x L(x_0, \lambda_0, \mu_0) = 0 \\ \lambda_0 \geq 0 \\ (\lambda_0)_j g_j(x_0) = 0 \text{ for } 1 \leq j \leq J \\ g(x_0) \leq 0 \\ h(x_0) = 0 \end{array} \tag{4.4.2}$$

If such a triple (x_0, λ_0, μ_0) exists, then x_0 is an optimum value for (4.4.1).

5. Interpolation and Approximation

❖

Lecture 10/28

Interpolation is the process of converting observational data (x_i, y_i) representing inputs and outputs to a function ϕ (and $\phi(x_i) = y_i$ and ϕ is reasonable in some sense). Polynomial interpolation is the special case where ϕ is assumed to be a polynomial. In harmonic analysis, typically ϕ is assumed to be a trigonometric polynomial (a truncated Fourier series). Polynomial interpolation is important for numerical approximations to ODE's, PDE's, and integrals by quadrature as well as data science. A related topic is deep learning, which approximates functions in very high dimensions using linear algebra and gradient descent.

5.1 Lagrange Polynomial interpolation ❖

Consider a set of $n+1$ points $\{(x_i, y_i)\}$ for $0 \le i \le n$. The polynomial interpolation problem is to find $\pi_m \in P_m = \{\text{space of degree } m \text{ polynomials}\}$ such that $\pi_m(x_i) = y_i$. If $n > m$, this is overdetermined; if $n < m$ this is underdetermined. If $n = m$, then:

Theorem 5.1. *With the setup above, if $n = m$ and the x_i are distinct, then there exists a unique $\pi_n \in P_n$ such that $\pi_n(x_i) = y_i$ for $0 \le i \le n$.*

Proof:

Define:

$$\ell_k(x) = \prod_{j=0, j\neq k}^{n} \frac{(x - x_j)}{(x_k - x_j)}$$

for $0 \le k \le n$. This has the property that $\ell_k(x_k) = 1$ and $\ell_k(x_i) = 0$ for $i \neq k$. This implies that $\ell_k(x)$ are linearly independent and hence form a basis of P_n. Now take $\pi_n(x) := \sum_{k=0}^{n} y_k \ell_k(x)$ satisfies our desired condition. For uniqueness, assume $\tilde{\pi}_n \in P_n$ also satisfies the desired condition. Therefore $(\pi_n - \tilde{\pi}_n)(x_i) = 0$. Since both polynomials have degree n and there are $n+1$ roots of $\pi_n - \tilde{\pi}_n$, by the fundamental theorem of algebra $\pi_n - \tilde{\pi}_n \equiv 0$.

□

Remark 5.2. The functions $\ell_k(x)$ are called *Lagrange* polynomials and π_n is called the Lagrange polynomial.

Theorem 5.3. *Suppose we have $n+1$ observed points $\{x_i\}$ and another point $x \in \mathbb{R}$. Let $I_x = [a, b] \subset \mathbb{R}$ with $x \in I_x$ and $x_i \in I_x$ for all i. Let $f \in C^{n+1}(I_x)$ and let π_n be the polynomial interpolant on $\{(x_i, f(x_i)\}$ constructed above. Then the error $E_n(x) := f(x) - \pi_n(x)$ is:*

$$E_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\omega_{n+1}(x)$$

where $\omega_{n+1}(x) = (x - x_0)\cdots(x - x_n)$ and $\xi(x)$ is some function on I_x.

Proof:

Define $G(t) = E_n(t) - \frac{\omega_{n+1}(t)}{\omega_{n+1}(x)} E_n(x)$ for fixed x as in the statement. We have $G(x_i) = 0$ for all i, by construction of E_n and ω_{n+1}. Moreover, $G(x) = 0$. Therefore $G(t)$ has $n+2$ zeros. Note also that since $f \in C^{n+1}(I_x)$, we also have $G(t) \in C^{n+1}(I_x)$. By the mean value theorem, $G'(t)$ has at least $n+1$ zeros. Continuing inductively $n+1$ times, we can see that $G^{(n+1)}$ has at least one zero. Let $\xi(x)$ be the zero of $G^{(n+1)}$ obtained by applying MVT $n+1$ times as we did. Therefore by construction, $G^{(n+1)}(\xi(x)) = 0$.

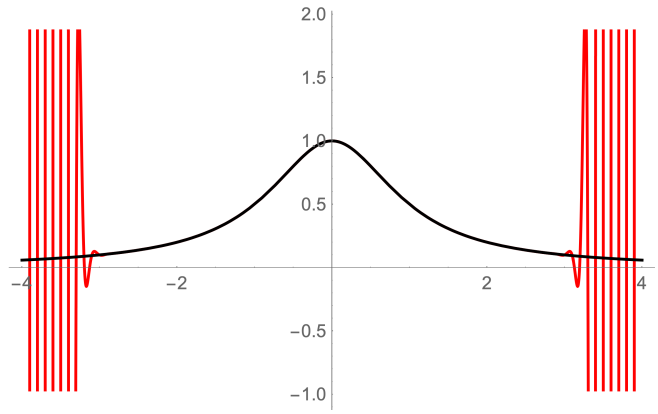


Figure 5.1: Runge's phenomenon for large degree polynomial interpolation.

We can calculate $G^{(n+1)}$ directly as:

$$\begin{aligned} G^{(n+1)}(t) &= E_n^{(n+1)}(t) - \frac{E_n(x)}{\omega_{n+1}(x)} \frac{d}{dt^{n+1}} \omega_{n+1}(t) \\ &= f^{(n+1)}(t) - \frac{E_n(x)}{\omega_{n+1}(x)} (n+1)! \end{aligned}$$

where we used that the $n+1$ derivative of π_n is zero. Now we plug in $t = \xi(x)$ and obtain the desired result.

□

There are instances where the error E_n can be really bad. For example, Runge's phenomenon; take $f(x) = \frac{1}{1+x^2}$, fix x_0 and set $x_i = x_{i-1} + \epsilon$ for $1 > \epsilon > 0$. Then the error E_n is very large near the end points (see Figure 5.1). This happens when the degree of the interpolated polynomial (i.e. number of points) is large and when the points are equally spaced.

Lecture 11/4

5.2 Piecewise Interpolation ❖

Equally spaced nodes do not guarantee uniform convergence of the interpolated polynomial when we take $n \to \infty$. This can be mended by taking piecewise interpolations. Make a partition T_h of the interval $[a, b]$ into k sub-intervals $I_j = [x_j, x_{j+1}]$ of length h_j such that $[a, b] = \bigcup_j I_j$. Set $h = \max_j h_j$. Now use Lagrange interpolation on each sub-interval I_j using $k+1$ equally spaced nodes x_j^i, where $0 \le i \le k$ for k sufficiently small. Note that different intervals can have different uniform spacing. For $k \ge 1$, define the spaces:

$$X_h^k = \{v \in C^0([a, b]) \ : \ v|_{I_j} \in P_k \ \ \forall I_j \in T_h\}$$

Then for any continuous function f on $[a, b]$, the piecewise interpolation polynomial $\pi_h^k \in X_h^k$ coincides on each I_j with the interpolation polynomial of $f|_{I_j}$ at the $k+1$ nodes $\{x_j^i\}$. Since the interpolation is done on each sub-interval with a small k, the polynomial interpolation will have less error than if it was done on the whole interval $[a, b]$ with a larger number of nodes. By Theorem 5.3, it follows that on each interval the error estimate is:

$$\|f|_{I_j} - \pi_h^k|_{I_j}\|_\infty \le C h^{k+1} \|f^{(k+1)}|_{I_j}\|_\infty$$

The following Theorem generalizes this estimate to the entire piecewise Lagrange interpolation.

Theorem 5.4. *Let $0 \leq m \leq k+1$ with $k \geq 1$ and assume that $f^{(m)} \in L^2([a,b])$. Then there exists a constant C independent of h such that:*

$$\|(f - \pi_h^k(f))^{(m)}\|_{L^2} \leq Ch^{k+1-m}\|f^{(k+1)}\|_{L^2}$$

where $\pi_h^k(f)$ is the piecewise Lagrange polynomial interpolation of f, done as described above. In particular, for $k = 1, m = 0$ or $m = 1$, we have

$$\|f - \pi_h^1(f)\|_{L^2} \leq C_1 h^2 \|f''\|_{L^2}$$
$$\|(f - \pi_h^1(f))'\|_{L^2} \leq C_2 h \|f''\|_{L^2}$$

Proof:

We will only prove the $m = 0, 1$ and $k = 1$ case. Let $e = f - \pi_h^1(f)$; note that $e(x_j) = 0$ for all $0 \leq j \leq k$. By Rolle's theorem, there exist $\xi_j \in (x_j, x_{j+1})$ for $j = 0, ..., k-1$ such that $e'(\xi_j) = 0$. By FTC, for $x \in I_j$, we have:

$$e'(x) = \int_{\xi_j}^{x} e''(s)ds$$

Therefore:

$$\begin{aligned} |e'(x)| &\leq \int_{\xi_j}^{x} |e''(s)|\, ds \leq \int_{x_j}^{x_{j+1}} |e''(s)|ds \\ &\leq \left(\int_{x_j}^{x_{j+1}} ds\right)^{1/2} \left(\int_{x_j}^{x_{j+1}} (e''(s))^2\, ds\right)^{1/2} \\ &\leq h^{1/2} \left(\int_{x_j}^{x_{j+1}} (f''(s))^2\, ds\right)^{1/2} \end{aligned} \tag{†}$$

Where we used Cauchy-Schwartz and that $\pi_h^1(f)$ is linear. Notice also that:

$$e(x) = \int_{x_j}^{x} e'(s)\, ds$$

So, using the inequality (†), we get:

$$|e(x)| \leq \int_{x_j}^{x_{j+1}} |e'(s)|ds \leq h^{3/2} \left(\int_{x_j}^{x_{j+1}} |f''(s)|^2 ds\right)^{1/2} \tag{†'}$$

Now we integrate the squares of the inequalities (†) and (†'):

$$\int_{x_j}^{x_{j+1}} |e'(x)|^2 dx \leq h^2 \int_{x_j}^{x_{j+1}} |f''(s)|^2\, ds$$

$$\int_{x_j}^{x_{j+1}} |e(x)|^2 dx \leq h^4 \int_{x_j}^{x_{j+1}} |f''(s)|^2\, ds$$

Finally, we obtain the result by summing over all j and taking the square root.

□

Lecture 11/6

5.3 Hermite-Birkhoff Interpolation ❖

The Hermite-Birkhoff interpolation is a generalization of the Lagrange interpolation where we also have data of derivatives. Suppose we are given data points $(x_i, f^{(k)}(x_i))$ where $i = 0, ..., n$ and $k = 0, ..., m_i$ for $m_i \in \mathbb{N}$. Let $N = \sum_{i=0}^{n}(m_i + 1)$. It can be shown that if the input nodes $\{x_i\}$ are distinct, then there exists a unique polynomial $H_{N-1} \in P_{N-1}$ called the *Hermite-Birkhoff* polynomial such that:

$$H_{N-1}^{(k)}(x_i) = y_i^{(k)}$$

where:

$$H_{N-1}(x) = \sum_{i=0}^{n}\sum_{k=0}^{m_i} y_i^{(k)} L_{ik}(x)$$

$$y_i^{(k)} = f^{(k)}(x_i)$$

and the L_{ik} are the Hermite characteristic polynomials, which have the property:

$$\frac{d^p}{dx^p}(L_{ik})(x_j) = \begin{cases} 1 & \text{if } i = j, p = k \\ 0 & \text{else} \end{cases}$$

The polynomials L_{ik} can be defined recursively. First, set $L_{im_i}(x) = \ell_{im_i}(x)$, where:

$$\ell_{ij}(x) = \frac{(x - x_i)^j}{j!} \prod_{k=0, k\neq i}^{n} \left(\frac{x - x_k}{x_i - x_k}\right)^{m_k+1}$$

Then let:

$$L_{ij}(x) = \ell_{ij}(x) - \sum_{k=j+1}^{m_i} \ell_{ij}^{(k)}(x_i) L_{ik}(x), \qquad j = m_i - 1, m_i - 2, ..., 0$$

Theorem 5.5. *We have the following estimate for the error in Hermite-Birkhoff interpolation:*

$$f(x) - H_{N-1}(x) = \frac{f^{(N)}(\xi)}{N!}\Omega_N(x)$$

where $\xi \in I_x$ and Ω_N is the polynomial:

$$\Omega_N(x) = (x - x_0)^{m_0+1}(x - x_1)^{m_1+1} \cdots (x - x_n)^{m_n+1}$$

Lecture 11/13

5.3.1 Polynomial interpolation in higher dimensions

In $\mathbb{R}^n$ for $n > 1$, there is no analogous existence theorem for polynomial interpolation as with $n = 1$. In particular, the geometry of how the points are arranged in $\mathbb{R}^n$ affects whether or not there is a unique solution (take, for example, three colinear points in $\mathbb{R}^2$ and fitting a linear function $\mathbb{R}^2 \to \mathbb{R}$ to them). However, there are ways around this if we restrict to certain types of problems:

a) Only considering polynomials in *tensor product form*:

$$P_\ell(x, y) = P_m(x)P_n(y)$$

b) Using predescribed shapes of points in $\mathbb{R}^n$ (e.g. nondegenerate simplices).

5.4 Splines ❖

While piecewise polynomial interpolation solves convergence issues, they can still produce non-differentiable approximations. The Hermite-Birkhoff method solves this issue by interpolating on derivatives; however, often the derivatives aren't known. In such cases, the method of splines is used. A *spline* is a piecewise polynomial with regularity constraints at the endpoints of the sub-intervals.

The most common type of splines are cubic splines $s(x)$, so that $s \in C^2([a,b])$ and $s(x)|_{[x_i,x_{i+1}]} \in P_3([x_i,x_{i+1}])$. We denote $s_i(x) = s(x)|_{[x_i,x_{i+1}]}$. They, of course, satisfy $s_i(x_i) = y_i$. The regularity conditions are:

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1})$$

$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1})$$

The continuity and regularity conditions by themselves don't impose enough conditions to get a unique solution. This is because there are $4n$ degrees of freedom, but only $4n-2$ equations. To fix this, we impose two derivative conditions on one or both endpoints of the entire interval. Typically, these are $s_0''(x_0) = 0$ and $s_0'(x_0) = m$ (these are called *natural* and *clamped*). Another possibility is periodicity: $s_0'(x_0) = s_n'(x_n)$ and $s_0''(x_0) = s_n''(x_n)$.

The resulting system of linear equations in the unknowns $\{a_{ij}\}$, $0 \le i \le n$ and $0 \le j \le 3$, which are the coefficients of s_i, can be solved using linear algebra in $\mathcal{O}(n^3)$ operations. However, it isn't immediately clear that the system is non-singular, nor is $\mathcal{O}(n^3)$ a very good complexity for practice. A way to rewrite the problem that makes the matrix easier to work with is to make the second derivatives $M_i = s''(x_i)$ the unknowns, which reduces the number of unknowns by a factor of 4 and will also make the resulting matrix tridiagonal. This is done by noting that $s_{i-1}''(x)$ is linear, and so we may write it as:

$$s_{i-1}''(x) = M_{i-1}\frac{x_i - x}{h_i} + M_i\frac{x - x_{i-1}}{h_i}$$

Where $h_i = x_i - x_{i-1}$. Integrating twice and imposing the endpoint conditions $s(x_{i-1}) = y_{i-1}$ and $s(x_i) = y_i$, we get:

$$s_{i-1}(x) = \frac{M_{i-1}}{6h_i}(x_i - x)^3 + \frac{M_i}{6h_i}(x - x_{i-1})^3 + \left(y_{i-1} - M_{i-1}\frac{h_i^2}{6}\right)(x - x_{i-1}) + \frac{y_i - y_{i-1}}{h_i} - \frac{h_i}{6}(M_i - M_{i-1})$$

So far, we have enforced the continuity conditions of s and s''. To get continuity of s', we must have:

$$\lim_{t\to 0^+} s'(x_i - t) = \lim_{t\to 0^-} s'(x_i - t)$$

We omit the calculation (see [Qa] §8.7.1) of these limits and just write the resulting equation:

$$\mu_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i \quad 1 \le i \le n-1 \tag{5.4.1}$$

where μ_i, λ_i, and d_i are constants defined by:

$$\mu_i = \frac{h_i}{h_i + h_{i+1}}$$

$$\lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}}$$

$$d_i = \frac{6}{h_i + h_{i+1}}\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right)$$

The $n-1$ equations in (5.4.1) have $n+1$ unknowns, so we must impose two more constraints on the endpoints of the form:

$$\begin{aligned} 2M_0 + \lambda_0 M_1 &= d_0 \\ \mu_n M_{n-1} + 2M_n &= d_n \end{aligned} \tag{5.4.2}$$

where μ_0, λ_0 are chosen in $[0,1]$ and d_0, d_n are arbitrary. To get a natural system ($s''(x_0) = s''(x_n) = 0$), we should choose $\lambda_0 = \mu_0 = d_0 = 0$. Another common choice is $\mu_0 = \lambda_0 = 1$ and $d_0 = d_1$ and $d_n = d_{n-1}$, which corresponds to extending the spline outside of $[a,b]$. In any case, we can write (5.4.1) and (5.4.2) as a square linear system:

$$\begin{bmatrix} 2 & \lambda_0 & 0 & \dots & 0 \\ \mu_1 & 2 & \lambda_1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \mu_{n-1} & 2 & \lambda_{n-1} \\ 0 & \dots & 0 & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

This is a tridiagonal nonsingular system and can be solved in $\mathcal{O}(n)$ operations using the Thomas algorithm. A general treatment of higher degree splines can be found in [Qa].

5.5 Application to Numerical Differentiation ❖

Lecture 11/18

An application of approximation by polynomials is numerical differentiation. If we have limited or impractical information about a function, we can instead approximate it by a much simpler piecewise polynomial and get simpler derivatives.

Example 5.6. Suppose we are given $f(x_0), f(x_1)$, we can approximate $f'(x_0)$. One way to do this is find a polynomial $\pi_n(x)$ that approximates f near x_0 (for example the Lagrange interpolant). The degree 1 interpolant, of course, will recover the usual approximation $(f(x_1)-f(x_0))/(x_1-x_0)$. The error formula we have for Lagrange polynomials

$$E_1(x) = f(x) - \pi_1(f)(x) = \frac{f^{(2)}(\xi(x))}{3!}(x-x_0)(x-x_1)$$
$$\implies E_1'(x) = f'(x) - \pi_1'(f)(x) = ☹$$

isn't helpful because we don't know how to differentiate $\xi(x)$. Instead we can use Taylor expansions to get the error. Let $x_1 - x_0 \equiv h$. Then:

$$\frac{f(x_0) - f(x_1)}{h} = \frac{f(x_0+h) - f(x_0)}{h} = f'(x_0) + \frac{h}{2} f''(\xi(x))$$

Therefore the error in the first order Lagrange approximation is $\frac{h}{2} f''(\xi(x))$.

Example 5.7. Now consider the same situation but instead we want $f''(x_0)$. The first order approximation is:

$$f''(x_0) \approx \frac{f(x_0-h) - 2f(x_0) + f(x_0+h)}{h^2}$$

Doing another Taylor expansion in h will give:

$$\frac{f(x_0-h) - 2f(x_0) + f(x_0+h)}{h^2} = f''(x_0) + \underbrace{\frac{h^2}{12} f^{(4)}(\xi(x))}_{\text{Error}}$$

We will see similar methods in more detail in the next section on numerical integration.

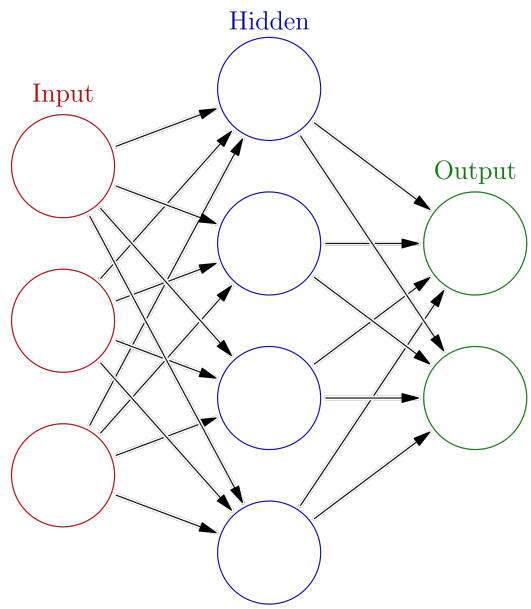


Figure 5.2: Artificial neural networks have input, output, and hidden layers. Source: Wikipedia

5.6 Deep Learning ❖

Lecture 11/11

A lot of the concepts in machine learning are old – at its most basic, it is a combination of linear algebra and optimization. The essential idea behind machine learning is to mimic human learning using artificial neural networks and reinforcement learning. While the ideas date back to the 1940's, it didn't find traction until recent computing power has allowed it to be effective.

A standard artificial neural network is typically represented as a weighted directed hypergraph where each node has a numerical value. In other words, it is a series of layers of nodes (neurons), with each layer has inputs from the previous and outputs into the next. The layers between the input (first) and output (last) layers are called hidden layers (see Figure 5.2). The edge weights of the inputs to a given node are linearly combined with the input node values and fed into an *activation function* to create the node's value. Some examples of activation functions are:

- ReLU (Relative Linear Unit): $ReLU(x) = \max(x, 0)$.
- Heaviside step function: $h(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$.
- Sigmoid: $\sigma(x) = \frac{e^x}{e^x+1}$.

In matrix notation, if $x^{(i)}$ is the ith layer as a vector, then $x^{(i+1)} = a_i(A_i x^{(i)} + b_i)$ where a_i is the activation function applied component-wise and A_i, b_i are an appropriate matrix/vector pair that encode the weights and shifts of the nodes. Typically, the matrices A_i are block matrices with many zeros.

5.6.1 Supervised Learning

The learning aspect of machine learning comes from optimizing the values of A_j, b_j to minimize some objective or cost function on the output nodes. Supervised learning is when we know how to classify the input and output

for the process of optimization (the data is labeled). Let $\{x^{(i)}\}_{0\leq i\leq k}$ be the layers and write the output nodes as $x^{(k)} = F(W, x^{(0)})$, where $W = \{A_0, b_0, A_1, b_1, ...\}$. Then, through unconstrained optimization, we wish to find:

$$W_{\text{optimal}} = \text{argmin}_W(C(W))$$

where $C(W) = \text{dist}(F(W, x^{(0)}) - \text{data})$. This is the distance (in some metric) between the output node values and the known (desired) outputs. The number of unknown parameters in W is typically very, very large, so optimization methods like simplex are not great. Sometimes gradient descent is used for larger dimensions. Recall this is defined by:

$$W^{(j+1)} = W^{(j)} - \alpha_j \nabla C(W^{(j)}) \tag{5.6.1}$$

where α_j is called the *learning rate*. However, even this is too costly for sufficiently high dimensions. In these cases, we instead use *stochastic gradient descent* (SGD). This consists of defining a cost function of the form:

$$C(W) = \sum_{\ell=1}^{K} C^{(\ell)}(W)$$

Then we perform gradient descent (5.6.1) on a random choice of sub-cost function $C^{(r)}(W)$, $1 \leq r \leq K$. The advantage here is that it is faster and can avoid local minima, due to the randomness.

The method of *backpropagation* evaluates $\nabla C^{(r)}(W)$ using the chain rule. Since $C^{(r)}$ contains $F(W) = a_k(a_{k-1}(....))$, it will have a large product of matrices. It is a fact that it is more efficient to evaluate $A(BC)$ (reverse multiplication) than $(AB)C$ (forward multiplication). Therefore, in evaluating the gradient, one should use reverse multiplication.

Remark 5.8. There is also a notion of unsupervised learning, which is where we don't have labeled data for optimization. In this case, the cost function is based on observable parameters (e.g winning/losing, entropy, etc) which serve as a heuristic for success. Two examples of such learning are adversarial learning and clustering.

However the optimization is done, convergence is guaranteed due to the following approximation theorem:

Theorem 5.9. *An artificial neural network can approximate a continuous function on a compact set with arbitrary accuracy.*

Lecture 11/25

5.7 Discrete approximation ❖

Sometimes we might be given data points or a collection of function values that we wish to interpolate but we don't necessarily want to use Lagrange polynomial interpolation. The simplest example of this is fitting a linear approximation to discrete data. This amounts to finding a and b such that $p_1(x) = ax + b$ minimizes the error. The error can be measured in any p-norm, but typically $p = 2$ (least squares) is chosen. Below we formulate the problem for various norms.

5.7.1 $p = \infty$

Suppose we are given points $(x_1, ..., x_m)$ and values $(f_1, ..., f_m)$. For the discrete approximation problem with $p = \infty$, we wish to find $p_n(x) = a_0 + a_1x + ... + a_nx^n$ such that $\|p_n - f\|_\infty := \max_j |p_n(x_j) - f_j|$ is minimal. This can be rewritten in matrix notation: $\|p_n - f\|_\infty = \|A\alpha - f\|_\infty$ (where A is the Vandermonde matrix associated to $(x_1, ..., x_n)$ and $\alpha = (a_0, ..., a_n)$). Thus we seek to find:

$$\min_\alpha \|A\alpha - f\|_\infty$$

We can convert this to a linear program by introducing a new variable $t = \|Ax - f\|_\infty$. Then the LP is:

$$\begin{gathered} \min\ t \text{ such that:} \\ -t\mathbf{1} \leq A\alpha - f \leq t\mathbf{1} \\ t \geq 0 \end{gathered}$$

where $\mathbf{1} = (1, ..., 1)$. The variables in this LP are (α, t).

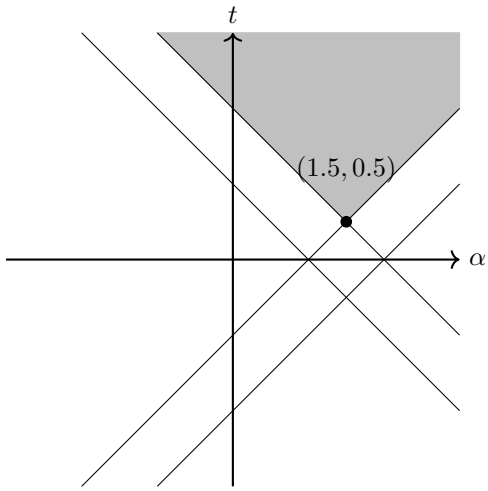


Figure 5.3: The region described by the inequalities in Example 5.10

Example 5.10. If we find $p_0 = a_0$, we only have one coefficent. Let our data be $f_1 = 1, f_2 = 2$ (the x_i are not necessary since we providing a constant approximation). The linear program becomes:

$$\begin{gathered} \min\ t \text{ such that:} \\ -t\begin{pmatrix}1\\1\end{pmatrix} \leq \begin{pmatrix}1\\1\end{pmatrix}\alpha - \begin{pmatrix}1\\2\end{pmatrix} \leq t\begin{pmatrix}1\\1\end{pmatrix} \\ t \geq 0 \end{gathered}$$

This can be solved pictorially (see Figure 5.3), and the solution is $(\alpha, t) = (1.5, 0.5)$. Therefore $a_0 = 1.5$.

Lecture 12/2

5.7.2 $p = 1$

Now suppose we work with $p = 1$; then we wish to minimize $\|A\alpha - f\|_1$. Letting $t = |A\alpha - f|$ (component-wise absolute value), we want to minimize $t^T\mathbf{1}$. Then the linear program becomes:

$$\begin{gathered} \min\ t^T\mathbf{1} \text{ such that:} \\ -t \leq A\alpha - f \leq t \\ t \geq 0 \end{gathered}$$

where t is now a variable along with α.

Example 5.11. Let $f_1 = 1$ and $f_2 = 2$ as in the previous example. The linear program is then:

$$\begin{gathered} \min\ t_1 + t_2 \text{ such that:} \\ -t_1 \leq \alpha_0 - 1 \leq t_1 \\ -t_2 \leq \alpha_0 - 2 \leq t_2 \\ t_1, t_2 \geq 0 \end{gathered}$$

It is left as an exercise to solve this LP.

5.7.3 $p = 2$

The case of $p = 2$ is the least squares solution to $\min_x \|Ax - b\|$. Recall in §2.7.1, we constructed the least squares solution using the SVD and the Morse-Penrose inverse $x = A^+b$.

5.8 Orthogonality and L^2 approximations ❖

In the case where we are approximating a function $f(x)$ continuously, we use the theory of orthogonal functions. This begins by assuming an approximation of the form:

$$f(x) \approx \sum_{j=0}^{J} \alpha_j \phi_j(x)$$

where $\phi_j(x)$ are functions of a special type (polynomial, exponential, wavelet). For example, when ϕ_j are exponentials, this is a truncated Fourier series. The problem is then finding the minimum:

$$\min_{\alpha} \left\| \sum_{j=1}^{J} \alpha_j \phi_j - f \right\|_2^2 = \min_{\alpha} \left(\sum_j \alpha_j \phi_j - f, \sum_k \alpha_k \phi_k - f \right)$$

where $(\cdot, \cdot)$ the L^2 inner product. This is a quadratic function in α, so we can take the gradient and set it to zero. Then setting the gradient to zero amounts to solving the equation:

$$A\alpha = f, \text{ where } A_{ij} = (\phi_i, \phi_j), f_j = (f, \phi_j) \tag{5.8.1}$$

Often the functions ϕ_j are chosen to be orthogonal so that A is diagonal. This allows us to directly solve the system 5.8.1 uniquely:

$$\alpha_j = \frac{(f, \phi_j)}{(\phi_j, \phi_j)}$$

A well-known result in Fourier theory is:

Proposition 5.12 (Parseval's Equality)**.** *Let ϕ_k be an orthonormal basis of L^2 and let $f \in L^2$. Then:*

$$\|f\|_2^2 = \sum_{k=1}^{\infty} \alpha_k^2 \|\phi_k\|_2^2$$

5.8.1 Orthogonal Polynomials

When ϕ_j are polynomials p_j, we need orthogonality. We do this by introducing a weight function $w(x)$ and an inner product:

$$(p_k, p_m)_w := \int_{-1}^{1} p_k(x) p_m(x) w(x)\, dx$$

For any weight function, we can build an orthogonal polynomial system. When $w = (1 - x^2)^{-1/2}$, the resulting orthogonal system is known as the Chebyshev polynomials; when $w = 1$, we get the Legendre polynomials. There are two other special inner products we define:

$$(p_k, p_m)_{\tilde{w}} := \int_0^{\infty} p_k(x) p_m(x) e^{-x}\, dx$$

$$(p_k, p_m)_{\bar{w}} := \int_{-\infty}^{\infty} p_k(x) p_m(x) e^{-x^2}\, dx$$

These have weights that suppress the singular behavior of polynomials at infinity. The resulting orthogonal systems from each of these are the Laguerre polynomials and the Hermite polynomials, respectively.

Example 5.13. The Chebyshev polynomials $T_k(x)$ can be found directly by the equations:

$$T_k(x) = \cos(k\theta)$$
$$\theta = \arccos(x)$$

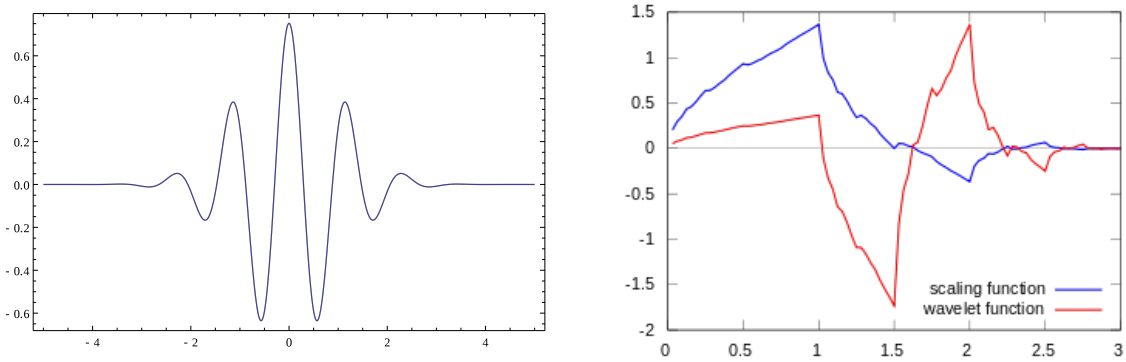


Figure 5.4: The Morlet wavelet (left) and Daubechies wavelet (right). Source: Wikipedia and Jon McLoone

We see $T_0 = 1$ and $T_1 = x$ by inspection. There is also a recursive formula via:

$$T_{k+1} = 2xT_k - T_{k-1}$$
$$T_0 = 1, T_1 = x$$

from which we find $T_3 = 2x^2 - 1$. Finally, there is a differential equation way to find $T_k(x)$:

$$(1 - x^2)T'' - xT' + \alpha^2 T = 0$$

Lecture 12/6

5.8.2 Wavelets and Signal Processing

Wavelets are an L^2 orthonormal basis $\{\psi_{n,k}(t)\}$ of mean zero functions parameterized by two integers. They are defined by:

$$\psi_{n,k}(t) = 2^{n/2}\psi(2^n t - k)$$

for a function $\psi(t)$ chosen so that they are orthogonal. This function is known as a *wavelet*. One such function introduced by Haar in 1909 for use in signal processing is the square wave:

$$\psi(t) = \begin{cases} -1 & 0 \leq t < \frac{1}{2} \\ 1 & \frac{1}{2} \leq t < 1 \\ 0 & \text{otherwise} \end{cases}$$

In 1983, Strömberg introduced a continuous wavelet with infinite support and C^p regularity. A more modern wavelet introduced by Morlet is:

$$\phi = \Re[C_1 e^{-C_2 t^2}\left(e^{iC_3 t} - C_4\right)]$$

This has infinite support and is C^∞. A C^p regular wavelet with finite support was introduced by Daubechies, but it had no simple expression nor was it symmetric (see Figure 5.4).

Wavelets find application in analysis and signal processing. They allow us to talk about frequencies at a point, which can't be done with Fourier series. In signal processing, wavelets allow us to perform what are known as importance sampling. This takes a representation of a signal $f = \sum \alpha_{n,k}\psi_{n,k}$ and approximates it as $\sum \tilde{\alpha}_{n,k}\psi_{n,k}$ where $\tilde{\alpha}_{n,k} = \alpha_{n,k}$ if $|\alpha_{n,k}|$ is larger than some tolerance and zero otherwise. This is the equivalent of a high pass filter. Importance sampling is also used for data compression, since it throws away information without affecting the signal too much. Doing the same, but for lower frequencies (a low pass wavelet filter), is useful for denoising. This is much more sensitive to local frequencies than the fourier low-pass filter.

One issue with wavelets is that they have the following property:

$$\int_{-\infty}^{\infty} \psi(t)\, dt = 0$$

Thus they can only approximate functions that integrate to 0 (they are only a basis of mean zero L^2 functions). To fix, this we introduce *shape functions*:

$$\phi_{n,k} = 2^{-n/2}\phi(2^n t - k)$$

for a function ϕ. These are not orthgonal, but they can be combined with wavelets to approximate all L^2 functions. Let $V_n = \mathrm{span}(\{\phi_{n,k}\}_{k\in\mathbb{Z}})$, a subspace of L^2; then we have a sequence of subspaces:

$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset L^2$$

Let $W_n = \mathrm{span}(\{\psi_{n,k}\}_{k\in\mathbb{Z}})$, the span of wavelets of a fixed n; then it can be shown that $V_{n+1} = V_n \oplus W_n$. This shows how a V_{n+1} function can be written as a sum of a V_n function and a wavelet. This is known as multiresolutional analysis.

6. Numerical Integration

❖

Lecture 11/18

6.1 Quadrature ❖

Just as we saw with numerical differentiation, we can use approximations to a function to calculate its integral. Let f be an integrable function and let $\pi_n(f)(x)$ be its Lagrange interpolant. Using this approximation for $n = 1, 2$ is called *quadrature*. For example, for $n = 1$, we have the approximation:

$$\begin{aligned}\int_a^b f(x)dx \approx \int_a^b \pi_1(f)(x)\, dx &= \int_a^b f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a}\, dx \\ &= \frac{f(a)+f(b)}{2}(b-a)\end{aligned}$$

This is the trapezoidal rule. The error can be computed with a Taylor expansion to be:

$$\begin{aligned}\int_a^b E_1(x)\, dx &= \int_a^b \frac{1}{6}f''(\xi(x))(x-a)(x-b)\, dx \\ &\leq \left|\frac{f''(\xi)}{6}\right| \int_a^b (x-a)(x-b)\, dx \\ &= \frac{-(b-a)^3}{12}|f''(\xi)|\end{aligned}$$

where ξ is a point in the interval $[a, b]$. The second inequality is valid since $(x-a)(x-b)$ does not change sign in the interval $[a, b]$. Using the approximation $\pi_2(f)$ results in:

$$\int_a^b f(x)\, dx \approx \frac{b-a}{6}\left[f(a) + 4f((a+b)/2) + f(b)\right]$$

This is Simpson's rule.

6.1.1 Composite methods

If we stack these approximations together (which corresponds to performing a piecewise interpolation on $[a, b]$), we get what are known as *composite* methods. These are what is used in practice. The composite trapezoidal rule comes from using the trapezoidal rule on piecewise linear approximations to f with step sizes h, m times. The resulting approximation is:

$$\int_a^b f(x)\, dx \approx \frac{h}{2}\left[f(a) + 2f(a+h) + 2f(a+2h) + ... + 2f(b-h) + f(b)\right]$$

Similarly, Simpson's rule has a composite version. Let $x_k = a + hk/2$ for $k = 0, ..., 2m$ be the end points of the subdivision of $[a, b]$ into $h/2$ sized sub-intervals. Then:

$$\int_a^b f(x)\, dx \approx \frac{H}{6}\left[f(a) + 2\sum_{r=1}^{m-1} f(x_{2r}) + 4\sum_{s=0}^{m-1} f(x_{2s+1}) + f(b)\right]$$

These are also known as the Newton-Cotes formulae (for general $\pi_n(f)$, not just linear and quadratic).

Lecture 11/20

6.2 Richardson Extrapolation ❖

We can leverage multiple approximations to an integral to obtain a more accurate approximation. This is known as Richardson interpolation. As an example, let $I(h)$ be an approximation to $I = \int_a^b f(x)\,dx$ (say, using composite trapezoidal) using intervals of width h and assume the error is given by an expansion in h as:

$$I(h) - I = c_1 h + c_2 h^2 + c_3 h^3 + \cdots$$

Then we also get:

$$I(h/2) - I = c_1 \frac{h}{2} + c_2 \frac{h^2}{4} + c_3 \frac{h^3}{16} + \cdots$$

If we truncate these expansions to first order in h, we get two equations with two unknowns (I and c_1). Then we can solve for c_1 and get a better approximation $I(h) - c_1 h$, which is $\mathcal{O}(h^2)$ to the actual integral. We denote this approximation by $I_{1,1}$. This can be done for any pair $I(h/2^{k-1})$ and $I(h/2^k)$ to get an $\mathcal{O}(h^2)$ approximation $I_{k,1}$. We can then take any pair of second order approximations $I_{k-1,1}$ and $I_{k,1}$ and combine them to get a $\mathcal{O}(h^3)$ approximation $I_{k,2}$ using the same steps, and so on. Diagrammatically, we represent this as:

$$\begin{tikzcd}
\mathcal{O}(h) & \mathcal{O}(h^2) & \mathcal{O}(h^3) & \dots \\
I(h) \arrow[dr] & & & \\
I\left(\frac{h}{2}\right) \arrow[r] \arrow[dr] & I_{1,1} \arrow[dr] & & \\
I\left(\frac{h}{4}\right) \arrow[r] \arrow[dr] & I_{2,1} \arrow[r] \arrow[dr] & I_{2,2} \arrow[dr] & \\
\vdots & I_{3,1} \arrow[r] & I_{3,2} \arrow[r] & \ddots
\end{tikzcd}$$

There is nothing special about the $\frac{1}{2}$ factor here; we could have just as easily used any $0 < \delta < 1$. If we set $I_{k,0} := I(h\delta^k)$, then the recursion relations describing the diagaram above can be worked out to be:

$$I_{m,q+1} = \frac{I_{m,q} - \delta^{q+1} I_{m-1,q}}{1 - \delta^{q+1}}, \quad q \geq 0 \text{ and } m > q$$

6.3 Singular Integrals ❖

A singular integral is one where there are singularities in the domain of integration but still $\int_a^b f(x)\,dx < \infty$. If we wish to approximate such an integral, we could split it up as:

$$\int_a^b f(x)\,dx \approx \int_a^{c-\epsilon} f(x)\,dx + \int_{c+\epsilon}^b f(x)\,dx$$

where $c \in [a, b]$ is the singular point. Then we could use quadrature for each of the above summands. Another technique is to find $g(x)$ such that $g(x) \sim f(x)$ near $x = c$ and $g(x)$ can has a known antiderivative. Then:

$$\int_a^b f(x)\,dx = \underbrace{\int_a^b (f(x) - g(x))\,dx}_{\text{Num. int}} + \underbrace{\int_a^b g(x)\,dx}_{\text{Exact}}$$

The function $f - g$ now has no singularities in $[a, b]$ and we can use quadrature to evaluate the first summand; meanwhile the second summand is known a-priori analytically. Often $g(x)$ can be chosen to be $\frac{1}{(x-c)^\mu}$ for $0 < \mu < 1$. The theory for this type of g is treated in detail in §9.8.2 of [Qa].

6.4 Higher Dimensions & Monte Carlo Methods ❖

Given a domain $\Omega \subset \mathbb{R}^2$, suppose we want to approximate a two dimensional integral

$$I(f) = \int_\Omega f(x,y)\, dx\, dy$$

Since Ω could be complicated, it would be nice if we could pick a bounding rectangle $R(\Omega)$ and extend f to $\tilde{f}$ by zero on $R(\Omega)$ and then compute $\int_{R(\Omega)} \tilde{f}(x,y)\, dx\, dy$. However, $\tilde{f}$ would generally not even be C^0, so quadrature methods will give very high errors. One way to fix this is to use stochastic (Monte Carlo) methods. Note that we have the equality:

$$\int_\Omega f(x,y)\, dx\, dy = |\Omega|\, \mathbb{E}[f]$$

where $\mathbb{E}[f]$ is the expected value of f in Ω. The expected value can be approximated as:

$$\mathbb{E}[f] \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n)$$

where $\{x_n\}$ are random points in Ω drawn uniformly. It is much easier to draw uniformly from $R(\Omega)$, so we instead use:

$$\int_\Omega f(x,y)\, dx\, dy = \int_{R(\Omega)} \tilde{f}(x,y)\, dx\, dy = |R(\Omega)|\, \mathbb{E}[\tilde{f}]$$

The area of the rectangle $R(\Omega)$ is easily calculated and so is an approximation to $\mathbb{E}[\tilde{f}]$. This method readily generalizes to arbitrary dimensions in the obvious way.

Remark 6.1. To generate random points in $R(\Omega)$, we use pseudo-random number generators. However, (pseudo) random numbers tend to cluster, so often *quasi-random* numbers are used instead. These are generated such that the numbers are approximately equally spaced in $R(\Omega)$.

Lecture 11/25

If the region $\Omega \subset \mathbb{R}^2$ is sufficiently nice, we can find a function $\phi : \Omega \to R$, where R is a rectangle, and use this transformation to convert the integral into an integral over a rectangular region:

$$\int_\Omega f(x)\, dx^n = \int_R f^*\phi(y)\, |\det(d\phi)|\, dy^n$$

If such a transformation can't be found, we could also slice up the region discretely and use a weighted sum. This gives an approximation of the form:

$$I(f) \approx h \sum_j \sum_k w_j w_{jk} \tilde{h}_j f(x_j, y_{k,j})$$

where w_j and w_{jk} are weights associated to each discrete point chosen in Ω. The method of Gaussian Quadrature adjusts these weights and discretization points for best accuracy. This generalizes to higher dimensions, but is only accurate for smaller dimensions, say 5 to 10. Any higher and Monte Carlo is a better option.

References

[Qa] Quarteroni, A., Sacco, R., Saleri, F. *Numerical Mathematics*. Texts in Applied Mathematics 37, Springer Verlag. Second Edition.

[Pa] Padberg, M. *Linear Optimization and Extensions*. Algorithms and Combinatorics, Springer. 2010.

[TB] Trefthen, L., Bau, D. *Numerical Linear Algebra*. Other Titles in Applied Mathematics, Vol. 50, SIAM, 1997.