Numerical Methods I

Olof Widlund
Transcribed by Ian Tobasco

Abstract. This is part one of a two semester course on numerical methods. The course was offered in Fall 2011 at the Courant Institute for Mathematical Sciences, a division of New York University. The primary text for the course will be *Numerical Linear Algebra* by L. Trefethen and D. Bau. *Analysis of Numerical Methods* by Isaacson and Keller may be helpful when we discuss orthogonal polynomials and Gaussian quadrature. There will be regular homeworks. The course website is `http://www.cs.nyu.edu/courses/fall11/CSCI-GA.2420-001/index.html`.

Contents

Chapter 1. Singular Value Decomposition and QR Factorization 5
1. Orthogonality 5
2. The Singular Value Decomposition 6
3. Projection Operators 9
4. The QR Factorization 10
5. Least-Squares 13

Chapter 2. Interpolation by Polynomials 17
1. Newton Interpolation 17
2. Hermite Polynomials 19
3. Interpolation Error 21
4. Piecewise Interpolation 22
5. Quadrature by Polynomial Interpolation 24
6. Orthogonal Polynomials 28
7. Gaussian Quadrature 30

Chapter 3. Solving Linear Equations 35

CHAPTER 1

Singular Value Decomposition and QR Factorization

Lecture 1, 9/8/11

What is this course really about? It's mainly about orthogonality and its uses.

1. Orthogonality

First, some notation. We consider $\mathbb{R}^n$ to be the linear space of column vectors $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$. We'll denote the transpose of x as $x^* = (x_1, \ldots, x_n)$. The (canonical) inner product will be

$$(x, y) = x^* y = \sum_{k=1}^{n} x_k y_k,$$

and as usual the (Euclidean) norm of x is

$$||x||_{\ell^2} = \sqrt{x^* x}.$$

Recall that vectors $v_1, \ldots, v_k \in \mathbb{R}^n$ are *orthogonal* if $v_j^* v_k = 0$ for $j \neq k$, and that a vector v is *normal* if $||v|| = 1$.

Why are orthogonal vectors important to scientific computing? Computers cannot do exact math. But we'll see that certain algorithms do not amplify the round-off errors so much: these methods will be known as "well-conditioned" and "stable". As it will turn out, orthogonality is key to produce well-conditioned methods. Orthogonality is also related to variational principles. A variational problem is one in which the answer is a maximizer/minimizer.

Proposition 1.1. *Suppose $x \neq 0$, then $\frac{d}{d\epsilon} ||x + \epsilon y|| \big|_{\epsilon=0} = 0$ iff $x^* y = 0$.*

Proof. Set $f(\epsilon) = ||x + \epsilon y||^2$. Then

$$\begin{aligned} f(\epsilon) &= (x + \epsilon y)^* (x + \epsilon y) \\ &= x^* x + 2\epsilon x^* y + \epsilon^2 y^* y, \end{aligned}$$

and so

$$f'(0) = 2x^* y.$$

Now

$$g(\epsilon) = \sqrt{f(\epsilon)} = ||x + \epsilon y||,$$

so

$$g'(0) = \frac{1}{2} \frac{1}{g(0)} f'(0) = \frac{x^* y}{||x||}.$$

Hence

$$\frac{d}{d\epsilon}||x+\epsilon y||\bigg|_{\epsilon=0} = \frac{x^* y}{||x||}$$

which gives the result. □

2. The Singular Value Decomposition

In this section we'll describe the SVD of a matrix. (In finance this is known as "principal component analysis", or PCA for short.) Let $A : \mathbb{R}^n \to \mathbb{R}^m$ be a $m \times n$ matrix. The SVD looks for the "important" parts of A. Here is an algorithm for computing the SVD of A.

Step 1. Compute $\sigma_1 = \max_{||v||=1} ||Av|| = ||A||$ and find v_1 such that $||v_1|| = 1$ and $||Av_1|| = \sigma_1$. (σ_1 represents the biggest "part" of A.) Also define u_1 via

$$Av_1 = \sigma_1 u_1$$

with $||u_1|| = 1$.

PROPOSITION 2.1. *If $v^* v_1 = 0$ and $\sigma_1 \neq 0$, then $(Av)^* u_1 = 0$.*

PROOF. By contradiction, suppose $v^* v_1 = 0$ but $w = Av$ has $u_1^* w \neq 0$. Then v_1 was not optimal. To see this, set

$$v(\epsilon) = \frac{v_1 + \epsilon v}{||v_1 + \epsilon v||}$$

and note that $||v(\epsilon)|| = 1$ for all ϵ. By hypothesis,

$$\frac{d}{d\epsilon}||v_1 + \epsilon v||\bigg|_{\epsilon=0} = 0$$

and thus

$$\frac{d}{d\epsilon}\frac{1}{||v_1 + \epsilon v||}\bigg|_{\epsilon=0} = 0$$

by the chain rule. Now let

$$\begin{aligned} w(\epsilon) &= Av(\epsilon) \\ &= \frac{1}{||v_1 + \epsilon v||}(Av_1 + \epsilon Av) \\ &= \frac{1}{||v_1 + \epsilon v||}(\sigma_1 u_1 + \epsilon w) \end{aligned}$$

where $w = Av$. Now observe that

$$\frac{d}{d\epsilon}||w(\epsilon)||\bigg|_{\epsilon=0} = 0$$

if v_1 is optimal. However,

$$||w(\epsilon)||^2 = \frac{1}{||v_1 + \epsilon v||^2}\left(\sigma_1^2 u_1^* u_1 + 2\epsilon\sigma_1 u_1^* w + \epsilon^2 w^* w\right)$$

and so

$$\frac{d}{d\epsilon}||w(\epsilon)||^2\bigg|_{\epsilon=0} = \frac{2\sigma_1 u_1^* w}{||v_1 + \epsilon v||^2}.$$

Thus

$$\frac{d}{d\epsilon}||w(\epsilon)||^2\bigg|_{\epsilon=0} = 0$$

only if $u_1^* w = 0$ (or $\sigma_1 = 0$, but we've assumed otherwise). □

EXERCISE 2.2. Find a non-calculus proof.

Step 2. Find the second most important vector. That is, compute

$$\sigma_2 = \max_{\substack{||v||=1 \\ v^* v_1 = 0}} ||Av||.$$

Note that $\sigma_2 \leq \sigma_1$. If v_2 is a maximizer, then set u_2 with $||u_2|| = 1$ via

$$Av_2 = \sigma_2 u_2.$$

Note that $||v_1|| = ||v_2|| = 1$ by choice, and the proposition gives that $u_1^* u_2 = 0$ as $v_1^* v_2 = 0$.

Step $\mathbf{k+1}$. Suppose we have orthonormal $v_1, \dots, v_k$ and $u_1, \dots, u_k$ along with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$. Then compute

$$\sigma_{k+1} = \max_{\substack{||v||=1 \\ v^* v_j = 0,\ j=1,\dots,k}} ||Av||$$

and if v_k is a maximizer find u_k with $||u_k|| = 1$ via

$$Av_k = \sigma_k u_k.$$

Another way of writing this is as follows. Define $S_k = \operatorname{span}\{v_1, \dots, v_k\}$, then

$$\sigma_{k+1} = \max_{\substack{||v||=1 \\ v \perp S_k}} ||Av||.$$

Again we have a proposition telling us how to proceed.

PROPOSITION 2.3. *Suppose $Av_{k+1} = \sigma_{k+1} u_{k+1}$ with $||u_{k+1}|| = 1$ as in the setup above. Then $u_{k+1}^* u_j = 0$ for $j = 1, \dots, k$.*

PROOF SKETCH. If $u_{k+1}^* u_j \neq 0$ then the v_j were not optimal. To see this, set

$$v_j(\epsilon) = \frac{v_j + \epsilon v_{k+1}}{||v_j + \epsilon v_{k+1}||}$$

for $j = 1, \dots, k$, and observe

$$\left. \frac{d}{d\epsilon} ||Av_j(\epsilon)|| \right|_{\epsilon=0} \neq 0$$

if $u_j^* u_{k+1} \neq 0$. □

NOTE. It could happen that $Av = 0$ if $v \perp S_k$. Then take $v_{k+1}, \dots, v_n$ to be arbitrary orthonormal vectors which are orthogonal to S_k, and set $\sigma_j = 0$ with $j \geq k$.

We've proved the following theorem.

THEOREM 2.4 (SVD). *Let A be a $m \times n$ matrix. Then there are orthonormal vectors $v_1, \dots, v_n$ and $u_1, \dots, u_n$ along with numbers $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ so that*

$$Av_k = \sigma_k u_k.$$

EXERCISE 2.5. Consider the integral transformation

$$f(x) = \int_0^L K(x, y)\, g(y)\, dy$$

with kernel $K(x,y) = \sin(x^2 y)$. Set $\Delta x = L/n$ and $x_k = k\Delta x + \Delta x/2$. Define an $n \times n$ matrix with components $A_{kj} = \Delta x \cdot K(x_k^2 y_j)$. Compute its singular values, and in particular compute $\log_{10}(\sigma_j)$. Observe the rate at which $\sigma_j \to 0$.

The theorem can be interpreted as a statement about matrix decomposition.

THEOREM 2.6 (SVD Decomposition). *Given a matrix A there are orthogonal matrices V and U along with a diagonal matrix Σ such that*

$$AV = U\Sigma,$$

i.e. such that

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

PROOF. Find u_k, v_k, σ_k as in the previous theorem. If $m > n$ then choose $u_{n+1}, \ldots, u_m$ to be orthonormal and orthogonal to $\operatorname{span}\{u_1, \ldots, u_n\}$. Set

$$V = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{pmatrix},$$

$$U = \begin{pmatrix} | & & | & | & & | \\ u_1 & \cdots & u_n & u_{n+1} & \cdots & u_m \\ | & & | & | & & | \end{pmatrix},$$

and

$$\Sigma = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & \\ & & \end{pmatrix}.$$

The proof is similar for $m < n$. □

What are the uses of SVD? Suppose

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

then

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

and hence

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

So V contains eigenvectors of A^*A and U contains eigenvectors of AA^*. And σ_k^2 are the non-zero eigenvalues.

EXERCISE 2.7. Show that the non-zero eigenvectors of A^*A and AA^* are the same.

Here is a second application: The low-rank approximation. Recall $\operatorname{rank}(A) = \dim(R(A))$. If A is rank 1, then $A = xy^*$ for some $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$. If A is

rank k, then $A = \sum_{j=1}^{k} x_j y_j^*$. The "best" rank k approximation to A is the matrix $\sum_{j=1}^{k} \sigma_j u_j v_j^*$. That is, if B has rank k, then

$$||A - B|| \geq ||A - \sum_{j=1}^{k} \sigma_j u_j v_j^*||.$$

There is a proof in the text.

Low-rank approximation allows for matrix compression. Indeed,

$$||A - \sum_{j=1}^{k} \sigma_j u_j v_j^*|| = \sigma_{k+1},$$

so when the $\sigma_k \to 0$ quickly, we can approximate the action of A with fewer and fewer numbers. Fewer numbers means fewer multiplies to compute Ax to a high degree of accuracy.

Note. The singular values and the eigenvalues of a matrix are very different. Indeed $\lambda_k \left(A^2\right) = \left(\lambda_k \left(A\right)\right)^2$ but $\sigma_k \left(A^2\right) \neq \left(\sigma_k \left(A\right)\right)^2$. To see how different they are, consider $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$. It's clear from this example that the singular values do not represent the dynamics (although the eigenvalues do).

Lecture 2, 9/15/11

3. Projection Operators

Definition 3.1. An $m \times m$ matrix P is a *projection* if $P^2 = P$.

For an arbitrary vector v we can write

$$v = Pv + (I - P)\,v.$$

This is a unique decomposition. Indeed, the subspaces $R\,(P)$ and $R\,(I - P)$ are complementary, in that $R\,(P) \cap R\,(I - P) = \{0\}$. (Suppose $Px = (I - P)\,y$, then applying P gives $Px = 0$ and hence the result.) So $\mathbb{C}^n = R\,(P) \oplus R\,(I - P)$. Note also that $I - P$ is also a projection, for

$$(I - P)^2 = I - 2P + P^2 = I - P$$

by definition. So we have decomposed the entire space into the direct sum of the rangespaces of (complementary) projections.

There is an important class of projections for which $R\,(P) \perp R\,(I - P)$. These are known as *orthogonal projections*.

Proposition 3.2. *A projection P is orthogonal iff $P = P^*$.*

Proof. Observe that

$$((I - P)\,v)^*\,Pv = v^*\,(I - P^*)\,Pv,$$

so if $P^* = P$ then P is an orthogonal projection.

For the other direction, recall that if we have $S_1 \oplus S_2 = \mathbb{C}^n$ then there exist orthonormal bases $\{q_1, \ldots, q_m\}$ and $\{q_{m+1}, \ldots, q_n\}$ for S_1 and S_2. Now take $R\,(P) = S_1$ and $R\,(I - P) = S_2$. Write $v = \sum_{i=1}^{n} (q_i^* v)\, q_i$ and apply P to get

$$Pv = \sum_{i=1}^{m} (q_i^* v)\, q_i = \sum_{i=1}^{m} q_i q_i^* v.$$

This shows $P = \sum_{i=1}^{m} q_i q_i^*$ and hence $P = P^*$. □

We can build an orthogonal projector P onto the columns of a given matrix. Suppose A is an $n \times m$ with full rank, and suppose $n > m$. Given a vector v, we want $(I - P)\, v \perp R(A)$. Call $Pv = Ax$ for some x, then this says $v - Ax \perp R(A)$. Thus

$$A^* (v - Ax) = 0$$

or just

$$A^* v = A^* Ax.$$

Now if

$$x^* A^* Ax = 0,$$

then

$$||Ax||^2 = 0$$

and hence $x = 0$. So $A^* A$ is invertible, and hence

$$x = (A^* A)^{-1} A^* v.$$

So what should Pv be? Applying A to both side we get

$$Pv = A\, (A^* A)^{-1} A^* v.$$

4. The QR Factorization

Suppose A is an $m \times n$ matrix. We look for a factorization

$$A = QR$$

with unitary Q and upper-triangular R. Such a factorization is called the *QR factorization* for A. Why should we care about QR? Suppose A is square, and that we want to solve the system

$$Ax = b.$$

Given the QR, we can write

$$\begin{aligned} QRx &= b \\ \implies Rx &= Q^* b \end{aligned}$$

which is a triangular system of equations. For example, consider a 2×2 triangular system of equations

$$\begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

Then

$$\begin{aligned} x_2 &= \frac{1}{r_{22}} c_2 \\ x_1 &= \frac{1}{r_{11}} (c_1 - r_{12} x_2)\,. \end{aligned}$$

This process generalizes immediately to the $n \times n$ case. And the work to solve a triangular system is exactly proportional to the number of non-zero entries.

4.1. Gram-Schmidt. Let's do a computation. Suppose q_1, q_2 are orthogonal unit vectors. Then we can write a projection $I - q_1 q_1^* - q_2 q_2^*$ which removes the components in the q_1- and q_2-directions. Now

$$I - q_1 q_1^* - q_2 q_2^* = \left(I - q_2 q_2^*\right)\left(I - q_1 q_1^*\right). \tag{4.1}$$

This observation gives us two ways in which to compute the QR of a matrix, the first of which is Gram-Schmidt. Suppose A is full rank and $n \times m$. Let the vectors $a_1, \ldots, a_m$ be the columns of A. The Gram-Schmidt process from basic linear algebra is

1. Set $q_1 = a_1/||a_1||$ and define r_{11} so that $a_1 = r_{11} q_1$.
2. Set $r_{12} = q_1^* a_2$ and $r_{22} = q_2^* a_2$. Set $a_2 = r_{12} q_1 + r_{22} q_2$.
3. Continue.

So we arrive at the decomposition

$$\begin{pmatrix} | & & | \\ a_1 & \cdots & a_m \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ q_1 & \cdots & q_m \\ | & & | \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ & r_{22} & \cdots & r_{2m} \\ & & \ddots & \vdots \\ & & & r_{mm} \end{pmatrix}.$$

Call

$$\hat{Q} = \begin{pmatrix} | & & | \\ q_1 & \cdots & q_m \\ | & & | \end{pmatrix}$$

and

$$\hat{R} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ & r_{22} & \cdots & r_{2m} \\ & & \ddots & \vdots \\ & & & r_{mm} \end{pmatrix},$$

then $A = \hat{Q}\hat{R}$ is called the *reduced QR decomposition* of A. Note that $\hat{Q}$ may not be unitary, as we could have $m < n$. But pick-up $q_{m+1}, \ldots, q_n$ which are orthogonal to $q_1, \ldots, q_m$ and of unit norm. Then define the $n \times n$ matrix

$$Q = \begin{pmatrix} & | & & | \\ \hat{Q} & q_{m+1} & \cdots & q_n \\ & | & & | \end{pmatrix}$$

and the $n \times m$ matrix

$$R = \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix},$$

then $A = QR$ is the *full QR decomposition.*

As it turns out, the algorithm prescribed above is numerically unstable. Instead, consider the following procedure, motivated by the second equality in equation 4.1:

1. Set $q_1 = a_1/||a_1||$.
2. Normalize $\left(I - q_1 q_1^*\right) a_2$ to obtain q_2.
3. Normalize $\left(I - q_2 q_2^*\right)\left(I - q_1 q_1^*\right) a_2$ to obtain q_3.
4. Continue.

This algorithm generates the same decomposition, but as it will turn out, it is numerically stable. We'll see why later.

4.2. Householder Transformations. There is a third way to find the QR factorization, via "Householder transformations". Suppose A is an $m \times n$ matrix with $m \geq n$. The idea is to produce unitary matrices Q_i such that $Q_n \cdots Q_1 A$ is upper triangular. How will we acheive this? Fix a vector v and consider the matrix $I - 2\frac{vv^*}{v^*v}$. This is not a projector, but observe

$$\left(I - 2\frac{vv^*}{v^*v}\right)^2 = I - 2\frac{vv^*}{v^*v} - 2\frac{vv^*}{v^*v} + 4\frac{vv^*vv^*}{(v^*v)^2} = I,$$

so it is unitary. Our goal is QR. So we should choose v so that

$$\left(I - 2\frac{vv^*}{v^*v}\right) a_1 = \begin{pmatrix} \pm||a_1|| \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Since $I - \frac{vv^*}{v^*v}$ is a projector, it's easy to see that $I - 2\frac{vv^*}{v^*v}$ reflects a_1 in the plane perpendicular to v (think of it geometrically). So we define Q_1 to be the unitary matrix $I - 2\frac{v_1v_1^*}{v_1^*v_1}$ with

$$v_1 = \begin{pmatrix} \pm||a_1|| - a_1 \\ -a_{21} \\ -a_{31} \\ \vdots \\ -a_{m1} \end{pmatrix} = \begin{pmatrix} \pm||a_1|| \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{pmatrix}.$$

We have yet to say how to choose the sign: it is best numerically to choose $+||a_1||$ if $a_{11} < 0$ and $-||a_1||$ if $a_{11} > 0$. (Subtracting two small numbers is numerically unstable.)

So now we have

$$Q_1 A = \begin{pmatrix} * & \\ 0 & \text{new} \\ \vdots & \text{variables} \\ 0 & \end{pmatrix}.$$

We want

$$Q_2 Q_1 A = \begin{pmatrix} * & * & & & \\ 0 & * & & & \\ \vdots & 0 & | & | & | \\ \vdots & \vdots & & & \\ 0 & 0 & & & \end{pmatrix},$$

so we set

$$Q_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & I - 2\frac{v_2v_2^*}{v_2^*v_2} & \\ 0 & & & \end{pmatrix}$$

for appropriate $v_2 \in \mathbb{C}^{n-1}$. And so on. After producing all the Q_i we have

$$Q_n \cdots Q_1 A = R$$

and hence

$$A = Q_1^* \cdots Q_n^* R.$$

Finally, consider solving $Ax = b$ for x. We have $Rx = Q^*b$ so we only need to compute Q^*b. We could first compute $Q^* = Q_n \cdots Q_1$ and then compute Q^*b. Will this save space on a computer? In fact the matrix-matrix multiplies required here will waste space. Instead, we should compute $Q_1 b$ then $Q_2(Q_1 b)$ then $Q_3(Q_2(Q_1 b))$ and so on. This step-by-step matrix-vector multiplication is much more efficient.

5. Least-Squares

We can apply the methods of this chapter to the age-old least-squares problem. Suppose we want to solve

$$Ax = b$$

with A an $m \times n$ matrix, $m > n$. From linear algebra, we know the best "solution" x will satisfy

$$A^*(Ax - b) = 0,$$

i.e.

$$A^*Ax = A^*b.$$

If A is of full rank, then

$$x = (A^*A)^{-1} A^*b.$$

Is there a more computationally efficient way to find x?

Lecture 3, 9/27/11

One way is to use QR. We saw three ways to compute the QR decomposition of a matrix A. The third method involved the Housholder transformation $I - 2\frac{vv^*}{v^*v}$, which reflects perpendicular to v. The first step was to set $x = A_1$ and $v = \operatorname{sgn}(x)||x||e_1 - x$, and then apply $I - 2\frac{vv^*}{v^*v}$ to A. Why did we choose to write $\operatorname{sgn}(x)$ in the definition of v? This prevents the possibility of subtracting two close numbers, which (as we'll see) would introduce a large amount of round-off error. To compute QR we also need to compute $||x|| = \sqrt{\sum x_i^2}$. Is it possible to do this accurately? Can we even compute $\sqrt{\alpha}$ accurately? Below, we discuss these issues; then, we'll solve least-squares.

5.1. Introduction to Error and Stability.

Computation introduces errors, and an important measure of an algorithm is the amount of relative error it introduces. Why are we interested in relative error? It gives us an idea of the number of reliable digits. A number is represented in binary form on a computer, in a so-called floating point system. For example, a number in the IEEE double format is a sequence $\pm\, a_1 \ldots a_{11}\ b_1 \ldots b_{52}$ where each entry is a one or zero. (See the handout for details.) Now suppose two numbers are stored in a given floating point system. All of the usual operations $+, -, \cdot, \div, \sqrt{}$ can be performed on these numbers, but the result may not be in the system. Of course, if the result is contained in the system then it is stored as such; otherwise, the computer chooses one of the closest numbers in the system to represent the result. Another way of saying this is that, in floating point, all the usual operations round to the last digit. We need a way to analyze the errors which are inherent to such a setup.

If x is a number, we denote the *floating point representation* of x as

$$(x)_{\mathrm{FL}} = (x)(1 + \epsilon)$$

where $|\epsilon| \sim 10^{-16}$. So the operation $+$ has as its floating point analogue the relation

$$(x+y)_{\mathrm{FL}} = (x+y)(1+\epsilon).$$

This carries over immediately to the rest of the usual operations. If $\epsilon = 0$ in this representation, then $x+y$ is in the floating point system. If $\epsilon \neq 0$, then the representation is not exact. This is exactly what leads to non-zero relative error: the relative error introduced by the operation of addition is the quantity

$$\text{relative error} = \frac{(x+y)_{\mathrm{FL}} - (x+y)}{(x+y)} = \epsilon.$$

What if we try to add three numbers? The relative error introduced would then be

$$\begin{aligned}
\text{relative error} &= \frac{((x+y)(1+\epsilon_1)+z)(1+\epsilon_2) - (x+y+z)}{x+y+z} \\
&= \frac{(x+y)\epsilon_1 + z\epsilon_2 + (x+y)\epsilon_1\epsilon_2}{x+y+z} \\
&\approx \frac{(x+y)\epsilon_1 + z\epsilon_2}{x+y+z}
\end{aligned}$$

since $\epsilon_1\epsilon_2$ will be small. Now observe that if we do not know the signs of x, y, z, then we cannot produce a bound on the error. But if x, y, z all share the same sign, the error will be at most $\epsilon_1 + \epsilon_2 + \epsilon_1\epsilon_2$.

What about a general function f? Even if $x + \Delta x$ is a representation of x with small error Δx, $f(x+\Delta x)$ may have large error relative to the expected result $f(x)$. The amount by which f magnifies the relative error in x is

$$\begin{aligned}
\text{error magnification} &= \frac{\left(\frac{f(x+\Delta x)-f(x)}{f(x)}\right)}{\left(\frac{\Delta x}{x}\right)} \\
&\approx \frac{f'(x)\cdot\Delta x}{f(x)\cdot\Delta x}\cdot x \\
&= \frac{f'(x)}{f(x)}\cdot x.
\end{aligned}$$

The smaller this magnification is, the better the resulting computation will be. If the error magnification associated with applying f is small, we say f is *numerically stable*.

Examples 5.1.

(1) $f(x) = \sqrt{x}$ has $\frac{f'(x)}{f(x)}x = \frac{1}{2}$ which is good. So $\sqrt{x}$ preserves relative errors (it does not magnify them), and hence is stable.
(2) Similarly, $||x||_2 = \sqrt{\sum x_i^2}$ is stable.
(3) $f(x) = e^x$ has $\frac{f'(x)}{f(x)}x = x$ which is not good for large x.
(4) $f(x) = 1 - x$ has $\frac{f'(x)}{f(x)}x = \frac{-x}{1-x} \to \infty$ as $x \nearrow 1$. This really shows that subtraction on a computer is unstable!

5.2. Solution of Least-Squares. Suppose A is a full rank matrix. The problem is to solve

$$Ax = b$$

as best as we can. If $b \in R(A)$ this is solvable exactly; but what if $b \notin R(A)$? Then the problem is to find x which minimizes $||Ax - b||_2^2$. Such an x is as close as we can get (in $||\cdot||_2$) to a solution of $Ax = b$.

Here is a first attempt at a solution method (the normal equations). Observe that x is a minimizer iff

$$A^* (Ax - b) = 0$$

since then $Ax - b$ will be perpendicular to the column space of A. Thus the minimizer is

$$x = (A^* A)^{-1} A^* b,$$

i.e. the unique solution to

$$A^* Ax = A^* b.$$

The advantage of this approach from a computing standpoint is that $A^* A$ is a small matrix in general. However, computing with $A^* A$ can be problematic due to round-off error. That is, even though A has full rank, the floating point version of $A^* A$ can be more singular.

There are other options, the most standard of which is via the QR-factorization. Let A be a $m \times n$ matrix with $m > n$, and suppose A has full rank. Then write $A = QR$ where

$$R = \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix}$$

with $\hat{R}$ a non-singular $n \times n$ matrix (A is of full rank). Then

$$||QRx - b||_2 = ||Rx - Q^* b||_2,$$

and if we write

$$Q^* b = \begin{pmatrix} c \\ \hat{c} \end{pmatrix}$$

with $c \in \mathbb{R}^n$ and $\hat{c} \in \mathbb{R}^{m-n}$, then the minimizer is exactly the solution of

$$\hat{R} x = c,$$

an upper triangular (read easily solvable) system of equations. We also get that the minimum error is $||Rx - Q^* b||_2 = ||\hat{c}||_2$. A clear advantage of this solution method is that it does not require computation of $A^* A$.

A second option is via the SVD-factorization. Then we have $A = U \Lambda V^*$, and

$$\begin{aligned} ||Ax - b||_2 &= ||U^* (U \Lambda V^* x - b)|| \\ &= ||\Lambda V^* x - U^* b||_2 \\ &= ||\Lambda y - U^* b||_2 \end{aligned}$$

once we call $y = V^* x$. Now if again

$$U^* b = \begin{pmatrix} c \\ \hat{c} \end{pmatrix}$$

with $c \in \mathbb{R}^n$ and $\hat{c} \in \mathbb{R}^{m-n}$, the minimizing y is the solution of

$$\Lambda y = c$$

which is trivial to solve, and then the minimizing x is exactly

$$x = V y.$$

The SVD often offers a fairly stable way to solve problems numerically.

5.3. Data Fitting. Suppose we have data (x_i, y_i) with $i = 1, \dots, m$. The problem is to find k, l so that $y = kx + l$ best approximates the data. More generally, the problem is to find $a_0, a_1, \dots, a_n$ so that $y = a_0 + a_1 x + \cdots + a_n x^n$ best approximates the data. If there are more measurements than unknowns, an exact fit is most likely impossible. But consider the system of equations

$$\begin{cases} a_0 + a_1 x_1 + \cdots + a_n x_1^n & = y_1 \\ & \vdots \\ a_0 + a_1 x_m + \cdots + a_n x_m^n & = y_m \end{cases}.$$

This can be written in matrix-vector form as

$$Ax = y$$

with

$$A = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & & & \\ 1 & x_m & \cdots & x_m^n \end{pmatrix},$$

$$x = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix},$$

and

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix},$$

Although A may not be invertible, it will be full-rank if $x_i \neq x_j$ for $i \neq j$. So the methods developed above will work to solve for the coefficients $a_0, \dots, a_n$ which give the best fit of the data.

(??) Here is a related problem. Suppose we want to solve $Ax = b$ but $N(A)$ is non-trivial. If A is of full-rank, then we can solve

$$AA^* y = b$$

exactly. Then the general solution to $Ax = b$ will be of the form

$$x = A^* y + z$$

for some $z \in N(A)$. And since $R(A^*) \perp N(A)$ we see that $A^{-1}(\{b\}) = R(A^*) \oplus N(A)$.

CHAPTER 2

Interpolation by Polynomials

An important observation in the development of numerical techniques is that many functions can be well-approximated by polynomials. The first step in this direction is polynomial interpolation. Given distinct points $x_0, x_1 \ldots, x_n$ and values $y_0, y_1, \ldots, y_n$, the problem is to identify an nth order polynomial $p_n(x) = a_0 + a_1 x + \cdots + a_n x^n$ which interpolates, i.e. which has $p_n(x_i) = y_i$ for $i = 0, \ldots, n$. This can be posed as a clssical matrix problem: find $a_0, \ldots, a_n$ which satisfy

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^n \\ \vdots & & & \\ \vdots & & & \\ 1 & x_n & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ \vdots \\ y_n \end{pmatrix}.$$

Note that the matrix on the left is a special type of matrix known as a "Vandermonde matrix".

How can we show there is a solution? One way is to look at the Vandermonde determinant. Or we can solve it directly by writing

$$p_n(x) = \sum_{i=0}^{n+1} y_i l_i(x)$$

with $l_i(x_j) = \delta_{ij}$. The l_is here are the *Lagrange polynomials*,

$$l_i(x) = \frac{\Pi_{k \neq i}(x - x_k)}{\Pi_{k \neq i}(x_i - x_k)}.$$

This method indirectly shows the invertibility of the Vandermonde matrix.

Thus there exists a solution. How should be find it numerically? We could try to invert the Vandermonde matrix, but this can be painful and often costs $o(n^3)$ time. Also, suppose we were to compute the interpolating $a_0, \ldots, a_n$ corresponding to data $(x_i, y_i)_{i=0,\ldots,n}$. If we add even a single data point, we would have to find all of the a_is over again. Can we solve this problem in a way which allows us to add additional data?

1. Newton Interpolation

Polynomials can always be rewritten around an arbitrary center point

$$p_n(x) = a_0^1 + a_1^1 (x - c) + \cdots + a_n^1 (x - c)^n.$$

This is just a shifted power series for p_n, but it is good to compute with when $|x-c|$ is small. In general, the *Newton form* of a polynomial is

$$\begin{aligned} p_n(x) &= a_0^{11} + a_1^{11}(x-c_1) + a_2^{11}(x-c_1)(x-c_2) + \cdots + a_n^{11}(x-c_1)\cdots(x-c_n) \\ &= a_0^{11} + (x-c_1)\left(a_1^{11} + (x-c_2)\left(a_2^{11} + a_3^{11}(x-c_3) + \dots\right)\right). \end{aligned}$$

This last line is the skeleton for fast polynomial evaluation. In this respect, the Newton form is superior to the Lagrange form of a polynomial. As we'll see next, the Newton form is also superior when it comes to interpolation.

Suppose we are given data $(x_i, f_i)_{i=0,\dots,n}$ and we want to find coefficients $A_0, A_1, \dots, A_n$ so that

$$p_n(x) = A_0 + A_1(x-x_0) + A_2(x-x_0)(x-x_1) + \cdots + A_n(x-x_0)\cdots(x-x_{n-1})$$

interpolates. Plugging in x_0 into the relation above yields $A_0 = f_0$; plugging in x_1 yields $A_1 = \frac{f_1-f_0}{x_1-x_0}$. In general we'll have

$$A_k = f[x_0,\dots,x_k] = \frac{f[x_1,\dots,x_k] - f[x_0,\dots,x_{k-1}]}{x_k - x_0}, \tag{1.1}$$

and so we call $A_k = f[x_0,\dots,x_k]$the kth *divided difference*. Computing A_k is an iterative process, as depicted in the table below.

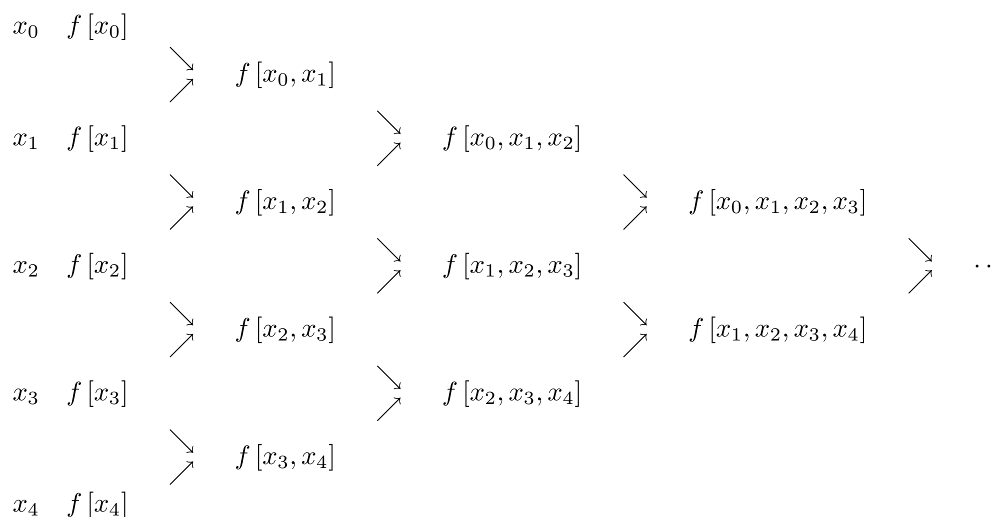

Table 1. Newton's triangular table.

How do we know the A_k satisfy (1.1)? Suppose $p_{k-1} \in P_{k-1}$ has $p_{k-1}(x_i) = f_i$ for $i = 0,\dots,k-1$ and $q_{k-1} \in P_{k-1}$ has $q_{k-1}(x_i) = f_i$ for $i = 1,\dots,k$. Then construct

$$p_k(x) = \frac{x-x_0}{x_k-x_0} q_{k-1}(x) + \frac{x_k - x}{x_k - x_0} p_{k-1}(x)$$

which satisfies $p_k(x_i) = f_i$ for $i = 0,\dots,k$. Now write

$$\begin{aligned} p_k(x) &= A_0 + \cdots + A_k x^k \\ p_{k-1}(x) &= B_0 + \cdots + B_{k-1} x^{k-1} \\ q_{k-1}(x) &= C_0 + \cdots + C_{k-1} x^{k-1} \end{aligned}$$

and track the leading coefficients to conclude

$$A_k = \frac{C_{k-1} - B_{k-1}}{x_k - x_0},$$

which is the result we're after. We have just developed the following result:

PROPOSITION 1.1. *The interpolating polynomial of* n*th degree is*

$$p_n(x) = \sum_{i=0}^{n} f[x_0, \ldots, x_i] \Pi_{j=0}^{i-1} (x - x_j).$$

In particular, the linear interpolant is

$$p_1(x) = f_0 + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0).$$

By construction, the quadratic interpolant p_2 includes p_1 in its description. More generally, p_n includes p_i for all $i < n$ in its description. So the formula above is recursive: it gives a way to systematically build up higher order interpolants of given data. Also, note that the ordering of the points $x_0, \ldots, x_n$ in the description above is irrelevant; interpolating polynomials are uniquely determined without regard to ordering of the data. This last observation can be easily turned into

PROPOSITION 1.2. *Divided differences are invariant under permutation of the data. That is, given any permutation* σ *on the set* $\{0, \ldots, n\}$,

$$f[x_0, \ldots, x_n] = f\left[x_{\sigma(0)}, \ldots, x_{\sigma(n)}\right].$$

2. Hermite Polynomials

Now suppose we are given data in the form (x_i, f_i, f_i'). Can we interpolate with a polynomial that matches both the desired function values and the corresponding derivatives? As a first example, consider the simple case $(x_i, f_i, f_i')_{i=0,1}$ where we are given only four pieces of data. For smooth enough f and small ϵ, this problem is close to the problem of interpolating the data set

$$\{(x_0, f(x_0)), (x_0 + \epsilon, f(x_0 + \epsilon)), (x_1, f(x_1)), (x_1 + \epsilon, f(x_1 + \epsilon))\},$$

which we would interpolate with

$$p_3^\epsilon(x) = A_0^\epsilon + A_1^\epsilon(x - x_0) + A_2^\epsilon(x - x_0)(x - (x_0 + \epsilon)) + A_3^\epsilon(x - x_0)(x - (x_0 + \epsilon))(x - x_1)$$

in the Newton scheme. The superscripts on the A_is indicate that the coefficients in the interpolation depend on ϵ. We can represent this interpolation with the following table:

Now we want to take $\epsilon \to 0$ and get an interpolant for the original data set $(x_i, f_i, f_i')_{i=0,1}$. Consider the divided difference

$$A_1(\epsilon) = f[x_0, x_0 + \epsilon] = \frac{f[x_0 + \epsilon] - f[x_0]}{\epsilon}.$$

As $\epsilon \to 0$, we recover $\lim_{\epsilon \to 0} A_1(\epsilon) = f_0'$. We can we make the suggestive labeling

$$f[x_0, x_0] = \lim_{\epsilon \to 0} \frac{f[x_0 + \epsilon] - f[x_0]}{\epsilon},$$

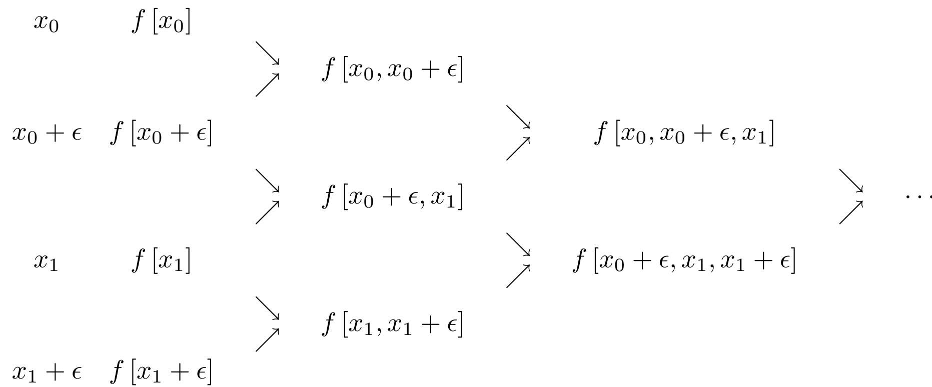

TABLE 2. Newton scheme with small ϵ.

then we have the relation $f[x_0, x_0] = f'(x_0)$. Similarly, $f[x_1, x_1] = f'(x_1)$. Now if we let $\epsilon \to 0$ in our interpolation, we recover

$$\begin{aligned} p_3^0(x) &= \lim_{\epsilon \to 0} p_3^\epsilon(x) \\ &= f[x_0] + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)(x - x_0) \\ &\quad + f[x_0, x_0, x_1, x_1](x - x_0)(x - x_0)(x - x_1). \end{aligned} \tag{2.1}$$

This is known as a *Hermite polynomial* and has the following table:

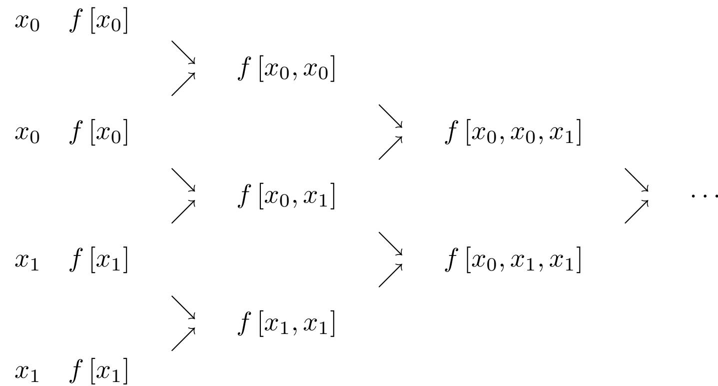

TABLE 3. Cubic Hermite polynomial.

Finally, we ask: does p_3^0 interpolate the given data $(x_i, f_i, f_i')_{i=0,1}$? It's clear from the table that $p_3^0(x_i) = f(x_i)$ for $i = 0, 1$. But also, we have

$$\frac{d}{dx}\left(p_3^0\right)(x_0) = f[x_0, x_0]$$

and

$$\frac{d}{dx}\left(p_3^0\right)(x_1) = f[x_1, x_1].$$

(The first is clear from (2.1); the second follows once we rewrite p_3^0 around the points x_1, x_1, x_0, for then the coefficient on $(x - x_1)$ will be $f[x_1, x_1]$. Both are

clear from the table.) Since $f[x_0, x_0] = f_0'$ and $f[x_1, x_1] = f_1'$, p_3^0 is the desired interpolating polynomial.

The discussion above showed how to arrive at cubic Hermite interpolation as a limit of approximating Newton schemes. This process generalizes immediately to arbitrary data sets of the form (x_i, f_i, f_i'), which are interpolated by higher order Hermite polynomials.

Lecture 4, 9/29/11

3. Interpolation Error

Recall the expansion

$$p_n(x) = \sum_{i=0}^{n} f[x_0, \ldots, x_i] \Pi_{j=0}^{i-1} (x - x_j)$$

for the nth order interpolating polynomial. This almost looks like a Taylor series, in that the coefficients are almost derivatives. So to estimate interpolation error, we proceed in a similar way as for Taylor series (where the nth order error is bounded in terms of the $(n+1)$th derivative).

Suppose p_n interpolates for f. Then the error is

$$e_n(x) = f(x) - p_n(x).$$

Given $\overline{x} \neq x_i$, consider the $(n+1)$th order interpolant

$$p_{n+1}(x) = p_n(x) + f[x_0, x_1, \ldots, x_n \overline{x}] \Pi_{j=0}^{n} (x - x_j).$$

In particular, this interpolates at $\overline{x}$ so that $p_{n+1}(\overline{x}) = f(\overline{x})$. Thus

$$\begin{aligned} e_n(\overline{x}) &= f(\overline{x}) - p_n(\overline{x}) \\ &= f[x_0, x_1, \ldots, x_n, \overline{x}] \Pi_{j=0}^{n} (\overline{x} - x_j) \end{aligned}$$

Is an exact expression for the nth interpolation error.

How can we tell if the error is big or small? As predicted above, the size of the nth divided difference for f depends on the higher derivatives of f. But also, the error depends on the distribution of the points x_i. Of course we cannot always choose x_i in practice, but if we can it would be advantageous to know the best points x_i to use (the so-called “Chebyshev points“).

Proposition 3.1. *The nth interpolation error satisfies*

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

for some ξ between the interpolating points.

This follows from

Theorem 3.2. *The kth divided difference satisfies*

$$f[x_0, \ldots, x_k] = \frac{f^{(k)}(\xi)}{k!}$$

for ξ between the smallest and largest of the points x_i.

Proof. For $k = 1$,

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(\xi)$$

for some ξ, by the mean value theorem. Now consider $e_k(x) = f(x) - p_k(x)$, which has $e_k(x_i) = 0$ for $i = 1, \ldots, k$. So $e_k'(\xi) = 0$ between the x_is. Thus, $f'(x) - p_k'(x)$

vanishes at at least k points. Similarly, $f''(x) - p_k''(x)$ vanishes at at least $k-1$ points. Going forwards, we see there must exist ξ so that $f^{(k)}(\xi) - p_k^{(k)}(\xi) = 0$. Now

$$\Pi_{j=0}^{k-1}(x - x_j) = x^k + \text{lower order terms}$$

and thus

$$f^{(k)}(\xi) = k! f[x_0, \ldots, x_k].$$

This proves the theorem. □

4. Piecewise Interpolation

Thus far, we have only discussed interpolation of data with a single polynomial. Now, we ask: when does a single polynomial fail to be a satisfactory interpolant? In practice, high order approximations turn out to be quite bad. One silly example can be found in MATLAB – the authors computed a high order polynomial interpolant for a large amount of censue data, and concluded negative population. A more classic example is due to Runge.

Example 4.1. Runge's phenomenon. Take $f(x) = \frac{1}{1+x^2}$ and interpolate on $[-5, 5]$ at equidistant points. Now increase the number of points and so the order of approximation. Observe how bad the approximation becomes!

But what if application demands interpolating at many points? Polynomials have the wonderful property of having infinitely many derivatives. But given a large data set, a better approach is to use piecewise polynomial interpolants. The simplest case is the piecewise linear, continuous approximation. This has the obvious disadvantage of kinks. In what follows, we'll develop a way to smooth out the kinks, via so-called "cubic splines". As we'll see, these are piecewise cubic, C^2 approximations to the data. But first, recall the cubic Hermite polynomials, built to match function values and first derivatives:

$$\begin{aligned} p_3(x) = f_0 + f'(x_0)(x - x_0) + \frac{f[x_0, x_1] - f'(x_0)}{x_1 - x_0}(x - x_0)^2 \\ + \frac{f'(x_0) - 2f[x_0, x_1] + f'(x_1)}{(x_1 - x_0)^2}(x - x_0)^2(x - x_1). \end{aligned}$$

This polynomial interpolates the data $(x_i, f(x_i), f'(x_i))_{i=0,1}$.

Now we build the cubic spline. If x_i is an interpolation point, we are forced to match $f(x_i)$; however, we are free to chose the derivative $s_i = f'(x_i)$. Since we desire a C^2 (piecewise) approximation, the obvious thing to do is the choose s_i so that second derivatives match at interpolation points. Explicitly, suppose we have two Hermite cubic polynomials which interpolate to the left and right of x_i. On the right we have

$$\begin{aligned} p_i(x) = f(x_i) + s_i(x - x_i) + \frac{f[x_i, x_{i+1}] - s_i}{x_{i+1} - x_i}(x - x_i)^2 \\ + \frac{s_i - 2f[x_i, x_{i+1}] + s_{i+1}}{(x_{i+1} - x_i)^2}(x - x_i)^2(x - x_{i+1}) \end{aligned}$$

and similarly on the left. Second derivative matching is $p_i''(x_i) = p_{i-1}''(x_i)$, and if we denote $\Delta x_i = x_{i+1} - x_i$ then (after some work) we arrive at the requirement

$$\Delta x_i s_{i-1} + 2(\Delta x_{i-1} + \Delta x_i) s_i + \Delta x_i s_i = 3(f[x_i, x_{i-1}]\Delta x_i + f[x_i, x_{i+1}]\Delta x_{i-1}).$$

Say $x_0, \ldots, x_n$ are the interpolating points, arranged in increasing order. Then at $x_1, \ldots, x_{n-1}$ we enforce the relation above. This is a system of $n-1$ equations. But there are $n+1$ unknowns, $s_0, s_1, \ldots, s_n$. The missing information is at the endpoints. If s_0, s_n are known a priori, then we'll have the same number of equations as unknowns and the the problem is solvable. Observe that the problem is non-local – there is coupling between neighboring intervals. But as the coupling itself is local, the problem is not hard to solve. To see why, consider the matrix equation corresponding to the system: it is of the form

$$\begin{pmatrix} * & * & & & \\ * & * & * & & \\ & & & & \\ & & * & * & * \\ & & & & \end{pmatrix} \begin{pmatrix} s_1 \\ \vdots \\ s_{n-1} \end{pmatrix} = \begin{pmatrix} \\ \vdots \\ \\ \end{pmatrix}.$$

This matrix is tridiagonal, and we claim it is always invertible. Calling the matrix J, suppose there exists y with $Jy = 0$. Then identify the largest component y_k of y, i.e. with $|y_k| \geq |y_i|$ for all i. Then

$$2\left(\Delta x_i + \Delta x_{i-1}\right) y_k = -\Delta x_i y_{k-1} - \Delta x_{i-1} y_k,$$

but

$$|-\Delta x_i y_{k-1} - \Delta x_{i-1} y_k| \leq |\Delta x_i + \Delta x_{i-1}||y_k|$$

which yields a contradiction unless $y_k = 0$. Such a matrix J is said to be *diagonally dominated*.

Now let's solve for the s_i. First, perform Gaussian elimination on J. The first step is

$$\begin{pmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & & \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{11} & a_{12} & & \\ 0 & \tilde{a}_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & & \end{pmatrix}$$

with $\tilde{a}_{22} = a_{22} - \frac{a_{12}a_{21}}{a_{11}}$. ($a_{11} \neq 0$ for a diagonally dominated matrix.) From this it's clear that after Gaussian elimination (relatively inexpensive computationally) we'll end up with an upper triangular system, which is efficiently solvable.

We have brushed over s_0 and s_n. Suppose, for example, that we don't know s_0. To determine s_0, we could require the spline to be C^3 (instead of just C^2) at the point x_1. This would yield s_0 in terms of s_1 and s_2. Or we could approximate s_0 with finite diferences. Or if we know a priori $f''(x_0) = f''(x_n) = 0$ (a physically legitimate boundary condition in some problems) then s_0 is automatically determined by s_1.

What about higher order splines? Doable, but cubics are much nicer to deal with in practice. What about interpolation in higher dimensions? This is an important question, and its solution has engineering applications (e.g. computer-aided design).

5. Quadrature by Polynomial Interpolation

This is an important application of interpolating polynomials. The goal is to compute

$$\int_a^b f(x)\ dx$$

as accurately as possible. The idea is the interpolate with a polynomial at a certain number of points, and then integrate the polynomial. And now that we can interpolate in a piecewise fashion, it will be sensible to write the integral above as the sum of integrals taken over sub-intervals. How then should we choose the points at which to break up the interval? It turns out this can be automated and optimized; we'll discuss so-called "adaptive quadrature" later in this section.

Our approximation is

$$\int_a^b f(x)\ dx \approx I(f) = \sum_{i=0}^{k} A_i f(x_i).$$

How can we determine the A_is? Let p_k be the kth order interpolant at points x_i, $i = 0, \ldots, k$, then

$$p_k(x) = \sum \Pi_{j=0, j\neq k}^{k} \frac{x - x_j}{x_k - x_j} f(x_k)$$

and so

$$A_i = \int_a^b \Pi \frac{x - x_j}{x_k - x_j}\ dx.$$

What error are we making in this approximation? We have

$$f(x) = p_k(x) + f[x_0, \ldots, x_k, x]\, \psi_k(x)$$

with $\psi_k(x) = \Pi_{j=0}^{k}(x - x_j)$. So the error is

$$E(f) = \int_a^b f[x_0, x_1, \ldots, x_k, x]\, \psi_k(x)\ dx.$$

Unfortunately, this is a quite complicated expression. Recall from calculus

$$f_{\min} \int_a^b g(x)\ dx \leq \int_a^b f(x)\, g(x)\ dx \leq f_{\max} \int_a^b g(x)\ dx$$

so long as g does not change sign on $[a, b]$. Then we can use an intermediate value argument to bound the integral. And as $f[x_0, \ldots, x_k, x] = f^{(k+1)}(\xi)/(k+1)!$ for ξ in $[a, b]$, the estimates will be explicit. In what follows, we present the results of this program for several different interpolants.

5.1. Trapezoid Rule. This first rule comes by interpolating f linearly at the endpoints of $[a, b]$. Let $\psi_1(x) = (x-a)(x-b)$ and note ψ_1 has non-zero integral on $[a, b]$. Also compute

$$\int_a^b (x-a)(x-b)\ dx = \frac{(b-a)^3}{6}$$

and

$$\int_a^b \left(f(a) + \frac{f(b) - f(a)}{b - a}(x - a) \right) dx = \frac{f(a) + f(b)}{2}(b - a).$$

Thus we have the *trapezoid rule*,

$$\int_a^b f(x)\,dx \approx \frac{f(a)+f(b)}{2}(b-a)$$

with associated error

$$\text{error} = -\frac{f''(\eta)}{12}(b-a)^3.$$

5.2. Midpoint Rule. Now, interpolate f linearly at the left endpoint and the midpoint of $[a,b]$. This yields the *midpoint rule*,

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

Let's evaluate the error

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

Since $x-(a+b)/2$ changes sign, we can't proceed directly as before. But note

$$\int_a^b \left(x-\frac{a+b}{2}\right)dx = 0,$$

and as

$$f\left[\frac{a+b}{2}, x\right] = f\left[\frac{a+b}{2}, \frac{a+b}{2}\right] + f\left[\frac{a+b}{2}, \frac{a+b}{2}, x\right]\left(x-\frac{a+b}{2}\right)^2$$

we find

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

Thus the error associated with the midpoint rule is

$$\text{error} = \frac{f''(\xi)}{24}(b-a)^3.$$

Note the midpoint rule is exact for both constant and linear functions. The next rule will be exact for quadratic functions, and also for cubic functions.

5.3. Simpson's Rule. Interpolate f with a quadratic polynomial at the endpoints and midpoint of $[a,b]$. Here are the details. We expect

$$\int_a^b f(t)\,dt = Af(a) + Bf\left(\frac{a+b}{2}\right) + Cf(b),$$

then A, B, C are determined via the canonical basis for third order polynomials, $\{1, t, t^2\}$. Of course we can use any basis, e.g. the Lagrange basis, to find the coefficients. Some are easier to work with than others.

Taking $f(t) \equiv 1$, we get

$$A + B + C = b - a.$$

Taking $f(t) = t-(a+b)/2$ we get $\int_a^b f = 0$, so

$$A\left(\frac{a-b}{2}\right) + C\left(\frac{b-a}{2}\right) = 0.$$

And for $f(t) = (t - (a+b)/2)^2$ we get $\int_a^b f = (b-a)^3/6$, so

$$A\left(\frac{b-a}{2}\right)^2 + C\left(\frac{b-a}{2}\right)^2 = \frac{(b-a)^3}{6}.$$

Solving these equations gives

$$A = C = \frac{1}{6}(b-a),$$
$$B = \frac{2}{3}(b-a).$$

Thus *Simpson's rule* is

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

The associated error is

$$\text{error} = -\frac{f^{(4)}(\xi)}{90 \cdot 2^5}(b-a)^5.$$

As remarked, Simpson's rule is exact up to cubic polynomials. This is an immediate consequence of the error estimate above; it's also easy to check each of $1, t-(a+b)/2, (t-(a+b)/2)^2, (t-(a+b)/2)^3$ are integrated exactly by the rule, and these form a basis for the cubic polynomials.

Lecture 5, 10/6/11

5.4. Corrected Trapezoid Rule. Interpolate with a Hermite cubic polynomial at the endpoints of $[a,b]$. This yields the *corrected trapezoid rule*,

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}(f(a) + f(b)) + \frac{(b-a)^2}{12}(f'(a) - f'(b)).$$

One could derive this by computation, or just check it is correct by checking exactness on cubic polynomials. The associated error is

$$\text{error} = \frac{f^{(4)}(\xi)}{720}(b-a)^5.$$

This rule can be useful when we have two subintervals $[x_i, x_{i+1}]$ and $[x_{i+1}, x_{i+2}]$ of the same width. The rule gives

$$\int_{x_{i+1}}^{x_{i+2}} f \approx \frac{x_{i+2} - x_{i+1}}{2}(f(x_{i+1}) + f(x_{i+2})) + \frac{(x_{i+2} - x_i)^2}{12}(f'(x_{i+1}) - f'(x_{i+2})),$$

and observe that the derivative term at x_{i+1} cancels upon adding this to the expression for $\int_{x_i}^{x_{i+1}} f$. So if evaluating f' is hard at some point $\tilde{x}$, by subdividing at $\tilde{x}$ and employing the corrected trapezoid rule twice, we can completely avoid evaluating $f'(\tilde{x})$.

5.5. Adaptive Simpson's Rule. The error estimate for Simpson's rule requires knowledge of the fourth derivative of f. But in practice, we don't know anything about the derivatives of f. So what good is this estimate? Suppose we have a function whose fourth derivative is well-behaved, and suppose we cut $[a,b]$ into subintervals and approximate the integral. If we then double the number of subintervals, how much better will the approximation become? This kind of question is immediately answerable via the error bound. (Suppose the fourth derivative is constant, then cut $[a,b]$ into two equal pieces and consider the error; divide those

pieces further and consider the error again.) This observation allows us to formulate a first algorithm for adaptive mesh refinement – the adaptive Simpson's rule.

Consider the integral

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\, dx$$

on a sub-interval $[x_i, x_{i+1}]$. Let $h = x_{i+1} - x_i$, then Simpson's rule says

$$I_i \approx S_i = \frac{h}{6}\left(f(x_i) + 4f\left(x_{i+\frac{1}{2}}\right) + f(x_{i+1})\right).$$

Consider dividing the subinterval further to get

$$\overline{S}_i = \frac{h}{12}\left(f(x_i) + 4f\left(x_{i+\frac{1}{4}}\right) + 2f\left(x_{i+\frac{1}{2}}\right) + 4f\left(x_{i+\frac{3}{4}}\right) + f(x_{i+1})\right).$$

Have we improved the approximation? We know

$$I_i - S_i = -\frac{1}{90} f^{(4)}(\eta)\left(\frac{h}{2}\right)^5$$

and

$$I_i - \overline{S}_i = -2\frac{1}{90} f^{(4)}(\overline{\eta})\left(\frac{h}{4}\right)^5.$$

(Think of $f^{(4)}(\overline{\eta})$ as an average over the two sub-subintervals.) So if $f^{(4)}$ is approximately constant, then

$$\overline{S}_i - S_i = \frac{f^{(4)} \cdot h^5}{2^5 \cdot 90}\left(\frac{1-2^4}{24}\right)$$

and so

$$I - \overline{S}_i = -\frac{\overline{S}_i - S_i}{15}.$$

Now suppose we want to evaluate the integral on the whole interval $[a, b]$ to within ϵ. Then if, after refining, we get

$$|I - \overline{S}_i| = \left|\frac{\overline{S}_i - S_i}{15}\right| \leq \frac{\epsilon h}{b-a},$$

adding the sub-subintervals was enough, and we should not refine any further. Conversely, if even after refining this criterion is not met, we should keep the sub-subintervals and try to refine each of them further. This is a recursive process.

When will such a process fail? Consider the problem

$$I(\lambda) = \int_a^b f(x, \lambda)\, dx$$

where we need to compute I as a function of λ. Adaptive mesh refinement may lead to vastly different meshes for different values of the parameter λ. But suppose I is smooth in λ a priori. Then this is a property we expect the computation to preserve. However, by introducing different meshes across different λs, we will easily fail to reproduce I as a smooth function after computing.

6. Orthogonal Polynomials

We have already seen one way of approximating an arbitrary function by polynomials, via polynomial interpolation. One can take a more abstract approach and view the space of real-valued functions as a normed linear space. Then any orthonormal basis produces readymade approximations in the given norm. In this section, we'll develop the basic theory of orthogonal polynomials and see some useful examples. In the next section, we'll apply the results to produce Gaussian quadrature.

Define a *weighted inner product* on the space of real-valued functions

$$(f,g)_\omega = \int_a^b \omega(x) f(x) g(x)\, dx$$

with the *weight* $\omega(x) \geq 0$ (ω must not vanish a.e.). As usual, $||f||_\omega^2 = (f,f)_\omega$. As we hope to approximate f, we look for polynomials which span this space; in particular we look for an orthonormal basis. Why do we care about finding such a basis? Given a function f, we hope to find the best possible approximation $p_n^* \in P_n$, satisfying

$$||f - p_n^*||_\omega \leq ||f - p_n||_\omega$$

for all $p_n \in P_n$. Suppose $\{\phi_i\}$ is an orthonormal basis, then all $p_n \in P_n$ are of the form $p_n = \sum_{i=0}^n \alpha_i \phi_i$. We can calculate explicitly

$$\Big|\Big|f - \sum_{i=0}^n \alpha_i \phi_i\Big|\Big|_\omega^2 = ||f||_2^2 - 2\sum_{i=0}^n \alpha_i (f,\phi_i)_\omega + \sum_{i=0}^n \alpha_i^2,$$

so to get the best approximation we should take $\alpha_i = (f,\phi_i)_\omega$. (One way to see this is to differentiate w.r.t. the α_i's and set the result equal to zero.) So an orthonormal set can be very convenient. The good news is there are plenty of accesible orthonormal bases: given any basis we can simply orthonormalize it via the Gram-Schmidt process.

Example 6.1. Take $w \equiv 1$, $a = -1$, $b = +1$. Start with the basis $\{1, x, x^2, \dots\}$ then orthonormalize with Gram-Schmidt. This produces the *Legendre polynomials*.

Proposition 6.2. *Suppose $\{\phi_j\}$ is an orthonormal set of polynomials under some weight ω and on $[a,b]$. Suppose $\phi_i(x) \in P_i$ is the ith degree polynomial in this set. Then all the roots of ϕ_i lie in (a,b).*

Proof. Define $Q_r = (x - x_1)\cdots(x - x_r)$ when $x_1, x_2, \dots, x_r$ are the roots of ϕ_i in (a,b). We want to prove $r = i$. Assume all roots are simple. Then

$$\int_a^b \omega(x)\phi_i(x) Q_r(x)\, dx \neq 0$$

for $\phi_i Q_r$ is of one sign on $[a,b]$. But if $r < i$ then $(\phi_i, Q_r) = 0$ which is a contradiction.

Can there be double roots? If x_0 is a double root then write $\phi_i(x) = (x - x_0)^2 \psi_{i-2}(x)$ and compute

$$0 = \int_a^b \omega(x)\phi_i(x)\psi_{i-2}(x)\, dx = \int_a^b \omega(x)\left[(x - x_0)\psi_{i-2}(x)\right]^2 dx \neq 0$$

which is a contradiction. This completes the proof. ∎

Although the Gram-Schmidt process is guaranteed to produce an orthonormal set out of any linearly independent set, the algorithm becomes less computationally tractable as the order of the polynomials goes up. Can we compute an orthonormal set $\{p_n\}$ in a more efficient manner? In fact, there is a recursive formula:

$$p_{n+1}(x) = (A_n x + B_n) p_n(x) - C_n p_{n-1}(x).$$

Of course, p_{n+1} is always describable in terms of $p_0, \ldots, p_n$, but what is interesting (and useful) here is that only p_{n-1} and p_n are needed? Let's prove this. For $k < n-1$ we have

$$\int_a^b \omega (A_n x + B_n) p_n(x) p_k(x)\, dx - \int_a^b \omega c_n p_{n-1}(x) p_k(x)\, dx = 0$$

since the p_j are orthogonal for $j \leq n$. So there are no additional terms needed. The coefficients A_n, B_n, C_n are determined in general by the system

$$\begin{cases} \int_a^b w (A_n x + B_n) p_n(x) p_n(x) = 0 \\ \int_a^b w (A_n x + B_n) p_n p_{n-1} = c_n \\ ||p_{n+1}||_w = 1 \end{cases}.$$

A_n, B_n, C_n are known for all of the classical polynomials.

6.1. Chebyshev Polynomials. Take

$$\omega(x) = \frac{1}{\sqrt{1-x^2}}$$

and $[a,b] = [-1,+1]$, then orthonormalize $1, x, x^2, \ldots$ by Gram-Schmidt. The result is the *Chebyshev polynomials*. The first polynomial is the constant $T_0(x) = \frac{1}{\sqrt{\pi}}$. In general,

$$T_n(x) = \sqrt{\frac{2}{\pi}} \cos(n \cdot \arccos x).$$

The recursion relation can be derived by trigonometry. From

$$\cos((n+1)\theta) + \cos((n-1)\theta) = 2\cos\theta \cos(n\theta)$$

we find

$$T_{n+1}(x) = 2x T_n(x) - T_{n-1}(x).$$

Once we know $T_0 = \frac{1}{\sqrt{\pi}}$ and $T_1 = \sqrt{\frac{2}{\pi}} x$ we can produce the rest via this relation.

The Chebyshev polynomials are quite interesting. For one, their roots (called the *Chebyshev points*) are very important for accurate interpolation and quadrature. The roots collect at the edges of $[-1,1]$. Now recall Runge's example. As it turns out, interpolating at the Chebyshev points is just the right thing to do to correct the error. And we'll see that the Chebyshev polynomials give the best approximation not only in the w-norm, but also in the ∞-norm.

Lecture 7, 8/20/11

6.2. Legendre Polynomials. Now take $\omega \equiv 1$, $[a,b] = [-1,+1]$, and orthonormalize $1, x, x^2, \ldots$ by Gram-Schmidt. This yields the *Legendre polynomials*, which are in general given by

$$L_n(x) = \sqrt{n + \frac{1}{2}} \frac{1}{n! 2^n} \left(\frac{d}{dx}\right)^n \left(x^2 - 1\right)^n.$$

Observe the degree of L_n is n. And the L_n are indeed orthogonal; to see this, write

$$\begin{aligned}(L_n, L_m) &= \int_{-1}^{1} L_n(x) L_m(x)\, dx \\ &= C \cdot \int_{-1}^{1} \left[\left(\frac{d}{dx}\right)^n (x^2-1)^n\right] \left[\left(\frac{d}{dx}\right)^m (x^2-1)^m\right] dx\end{aligned}$$

and integrate by parts repeatedly. They are also normalized, but we won't prove that here.

A related set of polynomials (sometimes also called the Legendre polynomials) are given by

$$l_n(x) = \frac{1}{\sqrt{n+1/2}} L_n(x).$$

These are not normalized, but instead satisfy $l_n(0) = 1$. The recurrance relation for the l_n is

$$l_{n+1}(x) = \frac{2n+1}{n+1} x l_n(x) - \frac{n}{n+1} l_{n-1}(x).$$

Now observe that

$$\frac{d}{dx}\left(\left(1-x^2\right)\frac{d}{dx}(l_n)\right) = \frac{d^2}{dx^2} l_n(x) - 2x\frac{d}{dx} l_n(x) = -n(n+1) l_n(x).$$

So the l_n are eigenvectors of the differential operator $\mathcal{L}(\cdot) = \frac{d}{dx}\left(\left(1-x^2\right)\frac{d}{dx}(\cdot)\right)$. Since

$$(\mathcal{L}f, g) = -\int_{-1}^{1} \left(1-x^2\right) f' g'\, dx = (f, \mathcal{L}g)$$

so $\mathcal{L}$ is self-adjoint. So its eigenvalues are all real; indeed, they are $-n(n+1)$.

We could go in the other direction, and start off by defining the operator $\mathcal{L}$ on the space of polynomials. Then if we look for a polynomial eigenvector $a_0 + a_1 x + \cdots + a_n x^n + a_{n+1} x^{n+1} + \ldots$ with eigenvalue $-n(n+1)$, we'll find $a_{n+1} = a_{n+2} = \cdots = 0$. So $-n(n+1)$ is exactly the right eigenvalue to demand to get an nth order polynomial eigenvector. Eventually if we solve for the $a_0, \ldots, a_n$, we'd end up at l_n.

6.3. Hermite Polynomials. What about unbounded domains? Take $a = -\infty$ and $b = +\infty$ and $\omega(x) = e^{-x^2}$. Orthonormalizing the standard basis now results in the *Hermite polynomials*. These take the form

$$p_n(x) = C_n \frac{1}{\omega(x)} \frac{d^n}{dx^n} e^{-x^2}$$

for some choice of constant C_n depending only on n. And as before, one can write down a differential equation that the p_n solve. One can do this for all the classical polynomials.

7. Gaussian Quadrature

Now we'll use the theory of orthogonal polynomials to approximate

$$\int_a^b f(x)\, dx.$$

The idea is still to approximate a function by a polynomial, and then to compute the integral of that polynomial. In other words, the goal is to find A_i so that

$$\int_a^b f(x)\, dx \approx \sum_{i=0}^n A_i f(x_i).$$

Simpson's rule is a choice of A_i which integrates polynomials of degree two exactly. And as it turned out, this rule also integrated polynomials of degree three exactly. Now we ask if it's possible to integrate higher degree polynomials exactly. We'll see that by selecting $A_0, \ldots, A_n$ and $x_0, \ldots, x_n$ in a clever way, we can integrate all of P_{2n+1} exactly. This is almost intuitive – consider that the vector space P_{2n+1} has dimension $2n+2$, and that there are $n+1$ points x_i and $n+1$ coefficients A_i.

7.1. Gauss-Lagrange Quadrature. We start by finding the Hermite interpolant of f at $x_0, \ldots, x_n$, which will match the data $\{f(x_i), f'(x_i)\}_{i=0,\ldots,n}$. As we are working towards quadrature, this may seem like a bad first step (we don't know $f'(x_k)$ a priori). But later we'll see that it's possible to choose the x_k so that the values $f'(x_k)$ are never needed. Now we guess that the Hermite interpolant takes the form

$$p(x) = \sum_{k=0}^n H_k(x) f(x_k) + \sum_{k=0}^n K_k(x) f'(x_k)$$

for some $H_k, K_k \in P_{2n+1}$. Observe that p will be the required interpolant if

$$H_k(x_i) = \delta_{ik},\ K_k(x_i) = 0,\ (H_k)'(x_i) = 0,\ (K_k)'(x_i) = \delta_{ik}$$

where δ_{ik} is the Kronecker delta. Recall the polynomials

$$L_k(x) = \frac{\Pi_{i=0,i\neq k}^n (x - x_i)}{\Pi_{i=0,i\neq k}^n (x_k - x_i)} \in P_n$$

used in Lagrange interpolation; these satisfy $L_k(x_i) = \delta_{ik}$. Define

$$\begin{aligned} H_k(x) &= (L_k(x))^2 \left(1 - 2\frac{d}{dx} L_k(x_k) \cdot (x - x_k)\right) \\ K_k(x) &= (L_k(x))^2 (x - x_k). \end{aligned}$$

We have $H_k(x_i) = \delta_{ik}$ and

$$\begin{aligned} \frac{d}{dx} H_k(x) = &\left(2L_k(x_k) \frac{d}{dx} L_k(x_k)\right)\left(1 - 2 \cdot \frac{d}{dx} L_k(x_k) \cdot (x - x_k)\right) \\ &+ (L_k(x))^2 \left(-2\frac{d}{dx} L_k(x_k)\right) \end{aligned}$$

so that $(H_k)'(x_i) = 0$ for all i. Also $K_k(x_i) = 0$ and

$$\frac{d}{dx} K_k(x) = 2L_k(x) \frac{d}{dx} L_k(x) \cdot (x - x_k) + (L_k(x))^2$$

so that $(K_k)'(x_i) = \delta_{ik}$. Thus we've procured the Hermite interpolant of f.

Now we'll integrate p to get the weights A_i. In doing so, we'll see how to choose the x_i. First, suppose $[a, b] = [-1, +1]$. Then write

$$\int_{-1}^1 p(x)\, dx = \sum W_k f(x_k) + \sum_{k=0}^n V_k f'(x_k)$$

with $W_k = \int_{-1}^{1} H_k(x)\, dx$ and $V_k = \int_{-1}^{1} K_k(x)\, dx$. If we can select $x_0, \ldots, x_n$ so that $V_k = 0$ for $k = 0, \ldots, n$, then the quadrature rule will read

$$\int_{-1}^{1} f(x)\, dx \approx \sum W_k f(x_k).$$

So observe

$$V_k = \int_{-1}^{1} L_k(x)\, L_k(x) \cdot (x - x_k)\, dx = C \int_{-1}^{1} L_k(x)\, \Pi_{i=0}^{n} (x - x_i)\, dx$$

for some constant $C \neq 0$. As $L_k \in P_n$, the idea is to choose $x_0, \ldots, x_n$ to be the roots of the $(n+1)$th Legendre polynomial $l_{n+1} \in P_{n+1}$. Then $\Pi_{i=0}^{n}(x - x_i)$ will be a multiple of l_{n+1} and hence $V_k = 0$ for $P_n \perp l_{n+1}$. (Recall the Legendre polynomials form an orthonormal basis on $[-1, 1]$ with $\omega \equiv 1$.) So we've found the desired interpolation points x_i and corresponding weights $A_i = W_i$. Note that $W_k > 0$ for all k; to see this compute

$$W_k = \int_{a}^{b} (L_k(x))^2\, dx - \int_{a}^{b} (L_k(x))^2 (x - x_k) \cdot 2\frac{d}{dx} L_k(x)\, dx$$

and observe that the second integral vanishes by our choice of x_i.

By a linear change of variable we can relax the assumption $[a, b] = [-1, 1]$. Then the general quadrature rule reads

$$\int_{a}^{b} f(x)\, dx \approx \frac{b-a}{2} \sum_{i=0}^{n} A_i f(\ell(x_i))$$

where $\ell(x) = \frac{b-a}{2} x + \frac{b-a}{2}$. This is called *Gauss-Lagrange quadrature*. The key idea was to approximate the integral of f by the integral of its Hermite interpolant at $n + 1$ quadrature points. As every polynomial $p \in P_{2n+1}$ can be exactly written as a Hermite interpolant at $n + 1$ distinct points, we can now integrate all of P_{2n+1} exactly.

7.2. Gauss-Lobatto Quadrature. Recall we proved that the roots of any polynomial belonging to an orthonormal set on $[-1, 1]$ must lie in the interior $(-1, 1)$. Thus the quadrature points in our scheme are in $(-1, 1)$. What if we require $x_0 = -1$, $x_n = +1$? In this case, there are $2n$ free parameters: $x_1, \ldots, x_{n-1}$ and $A_1, \ldots, A_{n-1}$. So we should expect to only integrate polynomials from P_{2n-1} exactly.

To derive the quadrature rule, we start by interpolating f with

$$p(x) = \sum_{k=0}^{n} H_k(x)\, f(x_k) + \sum_{k=1}^{n-1} K_k(x)\, f'(x_k)$$

where H_k, K_k are as above. Note the second sum does not include the endpoints. Again, we compute

$$\int_{-1}^{1} p(x)\, dx = \sum W_k f(x_k) + \sum_{k=0}^{n} V_k f'(x_k)$$

with $W_k = \int_{-1}^{1} H_k(x)\, dx$ and $V_k = \int_{-1}^{1} K_k(x)\, dx$. We hope to choose $x_1, \ldots, x_{n-1}$ so that $V_k = 0$ for all k. So far, the derivation has not changed.

Now, we have

$$V_k = C \int_{-1}^{1} L_k(x)\,(1-x^2)\Pi_{i=1}^{n-1}(x-x_i)\ dx$$

where $L_k \in P_n$. To ensure this was zero before, we chose $x_0, \ldots, x_n$ to be the roots of the $(n+1)$th Lagrange polynomial $l_{n+1} \in P_{n+1}$. But now observe that

$$\int_{-1}^{1} \frac{d}{dx} l_n(x) \frac{d}{dx} l_m(x) \left(1-x^2\right)\ dx = 0$$

for $n \neq m$, so that the derivatives $\frac{d}{dx} l_n$ of the Legendre polynomials form an orthgonal basis (up to normalization) with respect to the weight $\omega(x) = 1 - x^2$. With this in mind, we choose $x_1, \ldots, x_{n-1}$ to be the roots of $\frac{d^2}{dx^2} l_{n+1} \in P_{n-1}$, then

$$V_k = \tilde{C} \int_{-1}^{1} L_k(x) \left(\frac{d^2}{dx^2} l_{n+1}(x)\right) \omega(x)\, dx = 0$$

as $\frac{d^2}{dx^2} l_{n+1} \perp P_n$ with respect to ω. This is what we wanted.

To sum up, we have produced a quadrature rule at the points $-1 = x_0 < x_1 < \cdots < x_{n-1} < x_n = 1$ where $x_1, \ldots, x_{n-1}$ are the roots of the second derivative of the $(n+1)$th Legendre polynomial. This is known as *Gauss-Lobatto quadrature*.

What about $\int_{-\infty}^{\infty} f$?

CHAPTER 3

Solving Linear Equations

Consider what happens to $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ when we replace each a_i by $a_i + \delta a_i$. We ask: what happens to the roots? Why is this a relevant question? To get the eigenvalues of a matrix, we would solve for the roots of its characteristic polynomial. If we change the entries of the matrix, then the coefficients on the characteristic polynomial change accordingly. So the question is: how do the eigenvalues of a matrix change if its entries are perturbed?

Let's take $n = 20$, and say the roots are first $x_i = i$ with $i = 1, \ldots, 20$. As we change the coefficients, the roots x_i will change. We hope to understand this map. Recall we considered the ratio

$$|\frac{f(x+\delta x) - f(x)}{f(x)}|/|\frac{\delta x}{x}|$$

before with $f(x) = \sqrt{x}$ and so on. Now we look at this ratio with f the map from the coefficients to a particular root – the root x_{15}. Set $\tilde{p}(x) = \tilde{a}_0 + \tilde{a}_1 x + \ldots$ with $\tilde{a}_i = a_i + \delta a_i$ and look at the equations $p(x_j) = 0$ and $\tilde{p}(x_j + \delta x_j) = 0$. In particular suppose we perturb only the ith coefficient. Then we'll have $\tilde{p}(x) = p(x) + \delta a_i x^i$ and

$$\begin{aligned}
\tilde{p}(x_j - \delta x_j) &= p(x_j) + p'(x_j)\,\delta x_j + \delta a_i x_j \\
&= p'(x_j)\,\delta x_j + \delta a_i x_j \\
&= 0
\end{aligned}$$

to first order. Hence

$$\delta x_j = -\frac{\delta a_i x_j^i}{p'(x_j)}.$$

To get the denominator, write $p(x) = \Pi_{i=1}^{20}(x - i)$ and so $p'(x) = \sum_{j=1}^{20} \Pi_{i=1, i\neq j}^{20}(x - i)$. Thus $p'(x_j) = \Pi_{i=1, i\neq j}^{20}(x_j - x_i)$. It turns out the largest this gets is with $j = 15$, then $|p'(x_j)| = 4!15!$. The relative change in the roots is

$$\begin{aligned}
|\frac{\delta x_j}{x_j}|/|\frac{\delta a_i}{a_i}| &= \frac{|\delta a_{15}| 15^{14}}{4!15!} \cdot \frac{|a_i|}{|\delta a_{15}|} \\
&= \frac{15^{14}}{4!15!}|a_i|.
\end{aligned}$$

And $|a_i| \approx 1.16 \times 10^9$, so

$$|\frac{\delta x_j}{x_j}|/|\frac{\delta a_i}{a_i}| \approx 5.1 \times 10^{13}.$$

This is disasterous. What have we learned? Solving the characteristic polynomial to find the eigenvalues of a matrix is a bad idea.