

Notes for Math 211, Paris Center, winter 05

The topic of the course and the book is numerical analysis. This is the subject which studies algorithms for computing expressions which are defined with real numbers. It is an ancient subject. For example, one of the earliest tasks of interest was to compute square roots (and other roots), and Heron had a method to do that which we recognize now as a variant of the Newton-Raphson method.

Although the subject has ancient roots, it has had periods of intense development followed by long periods of stagnation. In many cases, the new developments have coincided with the introduction of new forms of computing machines. For example, many of the basic theorems about computing solutions of ordinary differential equations were proved right after desktop “adding machines” became common at the turn of the 20th century. The advent of the digital computer in the mid-20th century spurred interest in solving partial differential equations. However, many fundamental questions remain open, and the subject is an active area of research today.

The book is a classic in the field. It was written by experts in the subject who were working at one of the most active research centers at the time. Some topics are dated, but the level of presentation remains superb. Subsequent books have generally tried to present the subject in a less rigorous way.

1 Preliminaries, Chapter 1

1.1 Norms, Chapter 1, Section 1

The main point of the section is to introduce ways to estimate accurately the size of things which have complex forms. A *norm* provides such a measure. This is a simple generalization of Euclidean distance, but it can apply to rather complex objects such as operators on vector spaces.

Section 0 of Chapter 2 provides an introduction to (or review of) linear algebra concepts. This should be read first.

Since operators can be represented as matrices, one might think that it is sufficient just to have norms on Euclidean spaces. However, some operator norms cannot be written as a norm on a Euclidean space consisting of the coefficients of the corresponding matrix.

The “big” result of the section is that we can *almost* think of the spectral radius as a norm. That is, we can always find a vector norm such that the corresponding operator norm is arbitrarily close to the spectral radius.

Another result that is important is that all norms on a finite dimensional vector space are equivalent. Norms are also Lipschitz continuous.

Theorem 1 on page 2 presents a result similar to the Jordan decomposition of a matrix, but one that is less precise and much simpler to prove. The theorem says that any matrix is similar to a triangular matrix with diagonal entries given by the eigenvalues of the matrix. The Jordan canonical form is one such representation, however, we do not need such specific information. For us, it is also significant that the proof is very algorithmic, similar to algorithms we will consider later.

What sort of uniqueness holds for the decomposition in Theorem 1? Consider the following fact. Suppose that A and B are upper-triangular matrices of the form

$$A = \begin{pmatrix} a_1 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_2 & a_{23} & \cdots & a_{2n} \\ 0 & 0 & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & a_n \end{pmatrix} \quad B = \begin{pmatrix} b_1 & b_{12} & b_{13} & \cdots & b_{1n} \\ 0 & b_2 & b_{23} & \cdots & b_{2n} \\ 0 & 0 & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & b_n \end{pmatrix} \quad (1)$$

Then the product AB is (a) also an upper-triangular matrix with the property (b) that

$$AB = \begin{pmatrix} a_1b_1 & c_{12} & c_{13} & \cdots & c_{1n} \\ 0 & a_2b_2 & c_{23} & \cdots & c_{2n} \\ 0 & 0 & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & a_nb_n \end{pmatrix} \quad (2)$$

for some coefficients c_{ij} . That is, the diagonal entries of the product are the product of the diagonal entries.

One corollary of (2) tells us about the lack of uniqueness in Theorem 1. If $B = P^{-1}AP$ is upper-triangular, then for any upper-triangular matrix U with non-zero diagonal entries,

$$U^{-1}BU = U^{-1}P^{-1}APU = (PU)^{-1}A(PU) \quad (3)$$

provides another representation with the same properties stated in Theorem 1.

1.1.1 Convergent matrices, Chapter 1, Section 1.1

There are many situations in which the result of an algorithm can be written as multiplication by a fixed matrix A . Thus repeating the algorithm n times is equivalent to applying the matrix A^n . Sometimes, this might represent the error in some process. Thus we are interested in precise conditions when $A^n \rightarrow 0$ as $n \rightarrow \infty$.

Theorem 4 presents the main result giving three equivalent conditions that imply $A^n \rightarrow 0$ as $n \rightarrow \infty$. Its Corollary is a simple test which can be used as a guide. Theorem 5 uses the concept of convergence to give an algorithm for inverting a particular kind of matrix. Its corollary gives upper and lower bounds on the size of this inverse.

1.1.2 Homework

(1-3) page 16, numbers 1,2,5.

1.2 Floating point arithmetic, Chapter 1, Section 2

The main point of the section is to provide a way to analyze algorithms which are executed in floating point arithmetic on a digital computer. The algorithms we consider involve real numbers at an abstract level, but we are forced to work with a finite approximation of them.

The important thing to realize is that we are going to prove theorems about the actual computations, not just about their theoretical counterparts executed using real numbers. The latter is of interest, but not sufficient to guarantee success of the approximated computations done using floating point arithmetic. This point has not yet been fully understood in the field, and there have been other models of numerical analysis proposed, e.g., by Smale et al.

Actual floating point arithmetic is quite complex and does not lend itself to a simple representation. There is a floating point hardware standard developed by the IEEE and most hardware follows this at the moment. However, these specifications simply provide bounds on behavior of floating point.

The model used in the book is the standard way of representing an upper bound of the inaccuracies of floating point. We emphasize that is just a model, and there could be other more accurate models possible. One flaw of this model is that, even if some error bound is shown to be worst case for the model, it may not be worst case for floating point arithmetic for any set of data.

It could be that $f\ell(a+b) = f\ell(a) + f\ell(b)$ exactly for some a and b . However, it is also easy to see that this will fail in most cases. It might be an interesting exercise to compute the distribution of errors in floating point.

Cancellation is a major source of error. Note that $f\ell(a - b) = (a - b)(1 + \delta)$ does not mean that $f\ell(f\ell(a + e) - b)$ is at all close to $a + e - b$. *Floating point arithmetic is not associative*. Cancellation amplifies errors that have already occurred. It can easily be that $f\ell(a + e) = a$, and if $b = a$ we get zero for the result instead of e .

The analysis of computing an inner product in the section is an example of “backwards error analysis.” This technique, used by Wilkinson to study Gaussian elimination, provided a breakthrough in our understanding of numerical errors. Before that, the work of von Neumann and Goldstein had been a bit pessimistic about the prospects of solving large systems of equations.

The idea of backwards error analysis is to show that the following result holds for a given algorithm: for any input data, the approximate algorithm using floating point will produce the same result as the exact algorithm using real numbers but with a slightly perturbed input data set. Once this result is established, there is no need to worry further about the use of floating point. However, we still need to understand the *error propagation* of the algorithm: how does a small change in input data affect the output? This depends on the *stability* of the algorithm.

Backwards error analysis is itself still too pessimistic in important cases. One can give a fairly complete analysis in the case of computing a sum $\sum_{i=1}^n a_i$. Let A_n denote the result obtained by floating point arithmetic. Then

$$\sum_{i=1}^n a_i - A_n \approx \sum_{i=1}^n \sigma_i \delta_i + \mathcal{O}(\delta^2) \quad (4)$$

where $\sigma_j = \sum_{i=1}^j a_i$. Compare this with (9a) in the text which is much more pessimistic (but also a rigorous upper bound).

Doing the same level of analysis for more complex algorithms remains a topic of research.

1.2.1 Homework

(4) page 21, number 2.

(5) write a code to compute the slowly converging series

$$\frac{\pi}{4} \approx \sum_{i=1}^n \frac{(-1)^{i+1}}{2i - 1}$$

and compare with the error estimates the book suggests. Note that the order of computation is crucial. What is the worst order? The best order?

1.3 Well posed algorithms, Chapter 1, Section 3

This section attempts to provide a general framework for “well posed” algorithms. We won’t pursue the generality presented here but will rather just look at the particular example presented.

Suppose that we want to roots of a quadratic equation $x^2 + 2bx + c = 0$ where $b < 0$ and we chose the algorithm

$$x \leftarrow -b - \sqrt{b^2 - c} \quad (5)$$

This fails if we take $c = \epsilon^2 b^2$ since $x = 0$ as soon as $f\ell(1 - \epsilon^2) = 1$.

1.3.1 Homework

(6) page 25, number 1.

2 Matrix inversion/system solution, Chapter 2

2.1 Matrix examples

Many problems are posed using matrix equations. We give two examples to indicate the scale of the problem.

2.1.1 Two-point boundary value problems

Many physical models are described by differential equations. For a two-point boundary value problem, a finite difference approximation produces a linear system where the $N \times N$ matrix in question is of the form

$$\mathbf{A} = \begin{pmatrix} a_1 & b_1 & 0 & \dots & 0 & 0 \\ b_1 & a_2 & b_2 & \dots & 0 & 0 \\ 0 & b_2 & a_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & b_{N-2} & a_{N-1} & b_{N-1} \\ 0 & \dots & \dots & 0 & b_{N-1} & a_N \end{pmatrix} \quad (6)$$

where

$$b_i = \frac{-1}{h_i}, \quad a_i = \frac{1}{h_i} + \frac{1}{h_{i+1}} \quad (7)$$

for a mesh $x_0 < x_1 < \dots < x_N$ where $h_i = x_{i+1} - x_i$. Finding the deflection X of a rod subject to a force F would require solving $AX = F$.

Difference (or finite element) methods generate matrices with structure very similar to (6), with N limited only by the size of memory.

2.1.2 Implicit rankings

There are many situations in which it is desired to estimate rankings of related objects based on relationships among the objects. Suppose that we want to measure popularity. One way to do that is to see how many people keep your phone number in their personal digital assistants. If there are many such people, then you must be popular. But you are surely more popular if the people who keep your number are themselves popular. So a better way to rank things would be to keep track not only of how many people list you but also their rankings. For example, your ranking might be defined to be a simple multiple of the sum of all rankings of people who list you. However, this is a circular definition. How can you define your ranking until you know all other rankings? And how can you define them since presumably some of them must depend on your own?

The way out of this dilemma is to write the relationships using matrices and then observe that it leads to an eigenvalue problem, as follows. Let $\lambda > 0$ denote an unknown parameter for the moment which represents the constant of proportionality for rankings. Let $\mathbf{A} = (a_{ij})$ denote the matrix with the property that $a_{ij} = 1$ if and only if the j -th person lists the i -th person's phone number, and zero otherwise. Then the i -th ranking x_i can be determined from other rankings (x_j) by the relationship

$$x_i = \frac{1}{\lambda} \sum_{\{j : a_{ij} \neq 0\}} x_j. \quad (8)$$

This just says that the i -th ranking is proportional to the sum of the rankings of the entities that point to it, with constant of proportionality given by $1/\lambda$. Using properties of the definition of the matrix $\mathbf{A} = (a_{ij})$, we thus see that for all i

$$\lambda x_i = \sum_{\{j : a_{ij} \neq 0\}} x_j = \sum_j a_{ij} x_j = (\mathbf{A}X)_i. \quad (9)$$

This says that X and λ are eigenpairs: $\mathbf{A}X = \lambda X$.¹ Note that rankings, like eigenvalues, only can be meaningfully defined to within a constant factor. That is, absolute rankings make no sense; only rankings relative to others are useful.

Computing popularity of people based on phone number tabulation may be a silly example. But it is prototypical of other uses which are more realistic. The use of **link popularity** for search engines for the World Wide Web is an example. Instead of phone numbers, we record links to web pages. The above ideas carry over immediately and provide a model of rankings of web pages based on the links that point to them and their rankings. Current web search engines compute the corresponding eigenvalue problem for the entire web each day, with the order of a billion web pages ranked currently.

2.2 Gaussian elimination, Chapter 2, Section 1

The main point of the section is to formalize the familiar method used to solve systems of equations by elimination. The key point of Theorem 1 is that the “multipliers” m_{ij} form a matrix L which provides a factorization of $A = LU$. The matrix U is what is left at the end of elimination. The proof of this relation is by induction.

The factors can be used to solve problems with multiple right-hand sides by forward and backward substitution. This leads to an algorithm for computing the inverse of a matrix. The factorization also provides an algorithm to compute the determinant of a matrix. Operation counts for these are given in section 1.1.

One key point of the operation count section is that using the inverse matrix is not as efficient as solving equations.

By *pivoting* (i.e., ordering the equations and unknowns appropriately), we see that Gaussian elimination will allow us to find the rank of a matrix (Theorem 2). Compare this with recent research on computing the rank of a general tensor.

Uniqueness of the factorization: we see that $A = LDU$ gives a description of the general case, where both L and U are ones on the diagonal and D is a diagonal matrix. Factorization with pivoting just factors $LU = P^T A$.

2.2.1 Condition number, Chapter 2, Section 1.2

The condition number of a matrix A gives a way to show that solving a system of equations is well posed. The basic Theorem 3 is a worst-case analysis.

The backwards error analysis shows that matrix factorization in floating point may be viewed as giving a perturbed system. However, the bounds are quite pessimistic and a more complex analysis like the one we did for summation might be more appropriate.

Theorem 6 is due to Wilkinson and improved on more pessimistic estimates of von Neumann?

If we write $X(t)$ for the solution of

$$A(t)X(t) = F(t) \tag{10}$$

and differentiate we find

$$AX'(t) + A'X(t) = F' \tag{11}$$

or equivalently

$$X'(t) = A^{-1}F' - A^{-1}A'X(t) \tag{12}$$

and we take norms to find

$$\|X'(t)\| \leq \|A^{-1}F'\| + \|A^{-1}A'X(t)\| \tag{13}$$

¹If the matrix \mathbf{A} is irreducible [1], then there is a non-negative eigenpair, where λ is the dominant eigenvalue. In this case, the power method can be used to approximate X and λ .

which we simplify to get

$$\begin{aligned}
\frac{\|X'(t)\|}{\|X(t)\|} &\leq \frac{\|A^{-1}F'\|}{\|X(t)\|} + \frac{\|A^{-1}A'X(t)\|}{\|X(t)\|} \\
&\leq \frac{\|A^{-1}F'\|}{\|X(t)\|} + \|A^{-1}A'\| \\
&\leq \|A^{-1}\| \left(\frac{\|F'\|}{\|X(t)\|} + \|A'\| \right) \\
&= \kappa(A) \left(\frac{\|F'\|}{\|A\|\|X(t)\|} + \frac{\|A'\|}{\|A\|} \right) \\
&\leq \kappa(A) \left(\frac{\|F'\|}{\|F(t)\|} + \frac{\|A'\|}{\|A\|} \right)
\end{aligned} \tag{14}$$

Compare this with the estimate in Theorem 3, page 37 in the text.

2.2.2 A posteriori estimates and corrections, Chapter 2, Section 1.3

The residual error $R = F - AX$ left after applying some algorithm to “solve” $AX = F$ can (1) give an estimate of the error and (2) be used to correct it. More precisely, let $Y = f(A, F)$ be the result of such an algorithm, for which we know that (cf. Theorem 3, page 37)

$$\frac{\|Y - X\|}{\|X\|} \leq \epsilon \tag{15}$$

Then “solve” $E = f(A, R)$; $Y + E$ is often a better approximation to X . See section 4.3 for more details.

2.3 Direct factorization, Chapter 2, Sections 2 and 3

The Jordan “complete factorization” just combines LU and the forward and backward solves, so does not add anything new.

The key observation is that there is a simple algorithm to compute the $A = LU$ factorization once we know that it exists; see equations (4) and (5) on page 51. The number of floating point operations is the same, and the memory usage is also the same (can be done using only the storage allocated to A itself). However, the number of memory references are not the same. Note the phrase “reduce the number of intermediate quantities” on page 51. And also note the comment on “small storage” in the next sentence, which is misleading since the storage required can be the same. However, the comment about “hand computations” gives the right message.

Let’s do the numbers (for memory references). Well, I won’t repeat what we did in class, but just recall that the key estimates were

$$4 \sum_{\ell=1}^n \ell^2 = \frac{1}{3}n^3 + \mathcal{O}(n^2) \tag{16}$$

memory references for Gaussian elimination, and only

$$4 \left(\sum_{\ell=1}^n \ell(n - \ell) \right) + \mathcal{O}(n^2) = \frac{1}{6}n^3 + \mathcal{O}(n^2) \tag{17}$$

memory references for the direct factorization method. The factor of two improvement in memory references means a factor of two improvement in overall performance on contemporary computers.

Another advantage of the compact schemes is that higher precision can be used for the intermediary accumulations; see equations (4) and (5) on page 51.

The Cholesky factorization just uses the symmetry of A and writes $U = L^T$ which is one of the ways of combining LDU (take the square root of D). There is also a symmetric version of Gaussian elimination to compute the same factorization.

2.3.1 Homework

- (7) Prove that a triangular matrix is invertible if and only if its diagonal entries are nonzero.
- (8) page 49, number 4.
- (9) page 49, number 5.
- (10) page 49, number 6.
- (11) page 50, number 8 (prove lemma 1 on page 40).
- (12) page 61, number 1.

2.4 Iterative methods, Chapter 2, Section 4

The main point of the section is to consider approximate solution techniques with potentially fewer operations. The idea is a simple “splitting” of the matrix into $A = N - P$ where N is some matrix we are “willing” to invert. This could be a diagonal matrix or a triangular matrix. We solve $NX = F + PX$ by iterating $NX^{\nu+1} = F + PX^{\nu}$. The convergence theory for such methods is simple, iff $M = N^{-1}P$ is convergent.

Note the relationship $M = N^{-1}P = N^{-1}(N - A) = I - N^{-1}A$. Recall that when M is convergent, $I - M$ is invertible and $(I - M)^{-1} = M + M^2 + \dots$. Thus we are really computing $A^{-1}N = (N^{-1}A)^{-1} = M + M^2 + \dots$.

2.4.1 Jacobi method, Chapter 2, Section 4.1

The Jacobi method has N diagonal: $N = D$, where D is the diagonal of A . The matrix M_J for the Jacobi method also plays a role for Gauss-Seidel. The convergence estimate of the section says that when A is diagonally dominant, then Jacobi converges. It uses a simple representation of the ∞ and 1 norms as the maximal row and column sums.

2.4.2 Gauss-Seidel method, Chapter 2, Section 4.2

The Jacobi method has N diagonal and the Gauss-Seidel method has N a triangular matrix. The matrix M_J for the Jacobi method also plays a role for Gauss-Seidel. Lemma 1 in section 4.2 states that Gauss-Seidel converges when $\|M_J\|_{\infty} < 1$.

2.4.3 Residual correction, Chapter 2, Section 4.3

This section formalizes the idea of residual correction discussed earlier. Note that the matrix P in this section is the same as the matrix E in the backwards error estimates in Theorem 4 on page 41. We thus expect that $\|N^{-1}P\| \approx \|A^{-1}E\|$ is very small. The analysis here would imply that the residual would actually go to zero, but it is missing a key point that is taken up in section 1.1 of chapter 3.

2.4.4 Positive definite systems, Chapter 2, Section 4.4

Section 4.4 shows that the Jacobi and Gauss-Seidel methods are convergent for symmetric positive definite matrices. Think of the theorems this way. Suppose A is symmetric, positive definite. Then if $Q = N + N^t - A$ is positive definite, we get convergence.

For Gauss-Seidel, $Q = D$, where D is the diagonal of A . When A is positive definite, D is always positive.

For Jacobi, $Q = 2D - A$, where D is the diagonal of A . Suppose that $A = I + B$ for some matrix B . Then $Q = I - B$. Let the eigenvalues of B be denoted by λ_i^B . Then the eigenvalues of A are $1 + \lambda_i^B$ and those of Q are $1 - \lambda_i^B$. Thus the condition for Jacobi to be convergent is that

$$\min\{\lambda_i^B\} > -1 \quad (A > 0) \quad \text{and} \quad \max\{\lambda_i^B\} < 1 \quad (Q > 0) \quad (18)$$

The “ Q ” condition is clearly quite restrictive and indicates that Jacobi prefers diagonal dominance.

2.4.5 Homework

- (13) page 72, number 1.
- (14) page 72, number 2.
- (15) page 73, number 3.

3 Nonlinear equations, Chapter 3

We now turn to nonlinear systems of equations. We begin with one equation in one variable and later extend to systems. People have been trying to compute $\sqrt{2}$ for millenia:

<http://www.math.ubc.ca/people/faculty/cass/Euclid/ybc/analysis.html>

We will see that Heron's method

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{y}{x_n} \right) \quad (19)$$

can be viewed as Newton's method for computing \sqrt{y} .

3.1 Functional iteration, Chapter 3, Section 1

This goes by many names, including *fixed point iteration*, because it seeks to find a fixed point

$$\alpha = g(\alpha) \quad (20)$$

for a continuous function g . Fixed point iteration

$$x^n = g(x^{n-1}) \quad (21)$$

has the important property that, if it converges, it converges to a fixed point (20) (assuming only that g is continuous).

The rest of the story is then to figure out when and how fast it will converge. Basically, if g is Lipschitz continuous with constant $\lambda < 1$, that is,

$$|g(x) - g(y)| \leq \lambda|x - y| \quad (22)$$

then convergence will happen if we start close enough to α .

On the other hand, if $g'(\alpha) > 1$, it will diverge. In particular, the asymptotic behavior of fixed point iteration is given by

$$|x^n - \alpha| \approx Cg'(\alpha)^n \quad (23)$$

for some constant C .

It is instructive to consider the simple case

$$g(x) := \alpha + \lambda(x - \alpha) \quad (24)$$

where for simplicity we take $\lambda > 0$. Then for all n we have

$$|x^n - \alpha| = |g(x^{n-1}) - \alpha| = \lambda|x^{n-1} - \alpha| \quad (25)$$

and by induction

$$|x^n - \alpha| = \lambda^n|x^0 - \alpha| \quad (26)$$

where x^0 is our starting value. If $\lambda < 1$ this converges, but if $\lambda > 1$ this diverges.

3.1.1 Error propagation, Chapter 3, Section 1.1

Again, the affine function in (24) provides a good guide. Let us suppose that our computed function \hat{g} satisfies

$$\hat{g}(x) = g(x) + \delta(x) = \alpha + \lambda(x - \alpha) + \delta(x) \quad (27)$$

for some error function $\delta(x)$. If for example, we have $\delta(x) = \delta > 0$ for all x , then $\hat{g}(\hat{\alpha}) = \hat{\alpha}$ implies that

$$\hat{\alpha} = \hat{g}(\hat{\alpha}) = \alpha + \lambda(\hat{\alpha} - \alpha) + \delta = \alpha(1 - \lambda) + \lambda\hat{\alpha} + \delta \quad (28)$$

so that

$$\hat{\alpha} = \alpha + \frac{\delta}{1 - \lambda} \quad (29)$$

which shows that Theorem 3 on page 92 (see equation (18) there) can be sharp.

The effect of this kind of error on Newton's method is not so severe. As we calculated in class,

$$\hat{\alpha} - \alpha = \frac{\delta}{f'(\alpha)} \quad (30)$$

3.1.2 Second-order iterations, Chapter 3, Section 1.2

What happens if $g'(\alpha) = 0$? By Taylor's theorem

$$g(x) - \alpha = \frac{1}{2}(x - \alpha)^2 g''(\xi) \quad (31)$$

for some ξ and thus there error is squared at each iteration:

$$|e^n| = \frac{1}{2}(e^{n-1})^2 g''(\xi) \quad (32)$$

3.1.3 Homework

(16) page 96, number 1

(17) page 96, number 2.

3.2 Explicit methods, Chapter 3, Section 2

The chord method is a geometric method designed to point to a good place to look for the root:

$$x_{\nu+1} = x_{\nu} - \frac{f(x_{\nu})}{s}. \quad (33)$$

where $s = 1/m$ in the book's notation. This is based on the idea that the linear function ℓ with slope s which is equal to $f(x_{\nu})$ at x_{ν} vanishes at $x_{\nu+1}$ (so $\ell(x) = s(x_{\nu+1} - x)$). The chord method can be viewed as a damped version of the fixed point iteration where $g(x) = x - f(x)$. Here we take $g(x) = x - f(x)/s$. You can think of s as an adjustment parameter to help with convergence of the standard fixed point iteration.

Newton's method chooses the slope adaptively at each stage:

$$s = f'(x_{\nu}). \quad (34)$$

The chord method can be viewed as a type of difference approximation to this since we choose

$$s = \frac{0 - f(x_{\nu})}{x_{\nu+1} - x_{\nu}} \quad (35)$$

and we are making the approximation $f(x_{\nu+1}) \approx 0$ in defining the difference quotient.

The secant method (false position) approximates the slope by a difference method:

$$s = \frac{f(x_{\nu}) - f(x_{\nu-1})}{x_{\nu} - x_{\nu-1}}. \quad (36)$$

The error behavior is neither first nor second order, but rather

$$|e^n| = ce^{n-1}e^{n-2} \quad (37)$$

and repeating this says that $|e^n| \approx c|e^0|^{f_n}$ where f_n are the Fibonacci numbers. One iteration of secant only requires one function evaluation, so it can be more efficient than second-order methods.

Finally Steffensen uses an adaptive difference:

$$s = \frac{f(x_\nu + f(x_\nu)) - f(x_\nu)}{f(x_\nu)} \quad (38)$$

in which the usual $\Delta x = f(x_\nu)$ (which will go to zero: very clever).

The book gives a derivation of Steffensen via Aitken's process which is very general.

3.2.1 Homework

(18) page 108, number 1

(19) page 108, number 2

(20) page 108, number 3

(21) Show that Steffensen's method is second order convergent (hint: show that $g'(\alpha) = 0$ at the fixed point $\alpha = g(\alpha)$).

3.3 Functional iteration for systems, Chapter 3, Section 2

We just need to interpret the notation: g now maps R^n to itself, and fixed point iteration seeks to find a fixed point

$$\alpha = g(\alpha) \quad (39)$$

where now $\alpha \in R^n$. Fixed point iteration

$$x^\nu = g(x^{\nu-1}) \quad (40)$$

still has the property that, if it converges, it converges to a fixed point (39) (assuming only that g is continuous).

The rest of the story is similar, except that we need some higher-dimensional calculus to figure out when and how fast it will converge. Basically, if g is Lipschitz continuous with constant $\lambda < 1$, that is,

$$\|g(x) - g(y)\| \leq \lambda \|x - y\| \quad (41)$$

for some norm $\|\cdot\|$ on R^n , then convergence will happen if we start close enough to α .

3.3.1 Explicit methods, Chapter 3, Section 3.1

Not all of the one-dimensional methods generalize to n -dimensions. Suppose we want to solve $f(x) = 0$ where now f maps R^n to itself. The chord method becomes

$$x^{\nu+1} = x^\nu - Af(x^\nu). \quad (42)$$

where A is a matrix.

Newton's method takes $A = J_f(x^\nu)^{-1}$; that is, we solve

$$J_f(x^\nu)(x^{\nu+1} - x^\nu) = -Af(x^\nu). \quad (43)$$

3.3.2 Convergence, Chapter 3, Section 3.2

Convergence of Newton's method is again quadratic. The "abc" condition of Theorem 3 gives global conditions on convergence. Note that if $J_f(x^\nu)$ is nearly singular at any point, then the change $x^{\nu+1} - x^\nu$ can be huge. This occurs even in one dimension: consider $f(x) = \cos x$, and start Newton near $x = 0$.

3.3.3 Homework

- (22) page 123, number 3
- (23) page 123, number 4
- (24) page 123, number 5

4 Eigenvalue approximation, Chapter 4

We are skipping this.

5 Polynomial approximation, Chapter 5

5.1 Weierstrass and Bernstein, Chapter 5, Section 1

The Bernstein polynomial $B_n f$ can be used to prove Weierstrass' theorem. But the order of approximation is not optimal. Note that B_n is not a projection.

5.1.1 Homework

- (25) pages 186-7, number 1
- (26) Consider piecewise constant approximation on a uniform mesh of points i/n on $[0, 1]$. For a Lipschitz function, what is the best error estimate that you can give? Contrast this with the previous problem 25.

5.2 Lagrange interpolation, Chapter 5, Section 2

The existence of the Lagrange interpolant can be proved by constructing polynomials ϕ_i such that $\phi_i(x_j) = \delta_{ij}$. Then

$$Lf(x) = p(x) = \sum_{i=1}^n f(x_i)\phi_i \quad (44)$$

The operator L is a projection, since $p(x_i) = 0$ for $n+1$ points implies that a polynomial p of degree n must be identically zero. In fact, this approach can be used to show (problem 27) the existence of the ϕ_i 's.

The error in Lagrange interpolation satisfies

$$f(x) - Lf(x) = \frac{\prod_{i=0}^n (x - x_i)}{(n+1)!} f^{(n+1)}(\xi) = \frac{\omega_n(x)}{(n+1)!} f^{(n+1)}(\xi) \quad (45)$$

5.2.1 Homework

- (27) page 193, number 2

5.2.2 Chebyshev interpolation, Chapter 5, Section 4.2

For equally spaced points, the size of ω_n is quite large at the ends of the interval; see Figs 1a and 1b on pages 268 and 269. Chebyshev points

$$x_j = \frac{1}{2} \left(1 + \cos \left(\frac{j\pi}{n+1} \right) \right) \quad (46)$$

lead to the optimal $\omega_n(x) = \cos(n \cos^{-1} x)$. The main point is that the Chebyshev points are distributed quadratically near the end points of the interval.

One can prove that

$$\|f - L_n^c f\|_{max} \leq c(\log n) \inf_{p_n} \|f - p_n\|_{max} \quad (47)$$

Strangely, this is the best possible; there is no bounded linear projection onto polynomials. Note that the Bernstein operator is not a projection.

5.2.3 Homework

(28) page 229, number 1

5.2.4 Remainder term for Chebyshev and equidistant interpolation, Chapter 6, Section 3.1

This section computes precisely the differences in the error representation for Chebyshev and equidistant interpolation. It also has the Figs 1a and 1b on pages 268 and 269 which illustrate it.

5.2.5 Divergence of equidistant interpolation, Chapter 6, Section 3.4

This section estimates precisely the error for equidistant interpolation of $1/(1+x^2)$ on $[-5, 5]$ and shows that the error diverges.

5.3 Least squares, Chapter 5, Section 3

Another way to define approximations is by least squares. The basic polynomials are orthonormal:

$$(P_i, P_j) = \int_a^b P_i(x)P_j(x)w(x)dx = \delta_{ij} \quad (48)$$

where P_i is a polynomial of degree i of the form $P_i(x) = a_i x^i + q_i(x)$ where $a_i \neq 0$ and the degree of $q_i(x)$ is $i - 1$. This will be proved by induction.

This utilizes an inner-product structure on the linear space of square integrable functions. Then given any f define

$$L_n^S f = \sum_{i=0}^n (f, P_i) P_i. \quad (49)$$

Since the P_i 's are linearly independent, the set $\{P_0, \dots, P_n\}$ forms a basis for the space \mathcal{P}_n of polynomials of degree n .

Define the norm associated with the inner-product by

$$\|f\|_2 = \sqrt{(f, f)} = \sqrt{\int_a^b f(x)^2 w(x)dx} \quad (50)$$

and we can see that

$$\|f - L_n^S f\|_2 = \min_{q \in \mathcal{P}_n} \|f - q\|_2 \quad (51)$$

Just note that

$$(f - L_n^S f, q) = 0 \quad (52)$$

for all q of degree n .

Suppose that $q \in \mathcal{P}_n$. Then $q = L_n^S q$ (L_n^S is a projection) because we must have $\|q - L_n^S q\| = 0$.

Moreover

$$\|f - L_n^S f\|_2 = 0 \quad (53)$$

if and only if $f \in \mathcal{P}_n$.

The polynomials can be defined by induction (called the Gram-Schmidt process, see section 3.1 of chapter 5): $P_0(x) = 1/(b - a)$ and

$$P_n = \frac{1}{\|x^n - L_{n-1}^S x^n\|_2} (x^n - L_{n-1}^S x^n) \quad (54)$$

Observe the following facts about P_n . All of the roots of P_n are in the interval $[a, b]$. To see this, enumerate all such roots with multiplicity as x_0, \dots, x_m ; let $q(x) = (x - x_0)(x - x_1) \cdots (x - x_m)$. Then $r = P_n q$ is of one sign in $[a, b]$. Thus $(P_n, q) > 0$. Since P_n is orthogonal to \mathcal{P}_{n-1} , we must have $m = n$.

Also, we claim that the roots of P_n are all simple. For suppose that $P_n(x) = (x - x_1)^2 r(x)$. Then $P_n(x)r(x) = (x - x_1)^2 r(x)^2$ and we reach a contradiction that $(P_n, r) > 0$.

5.3.1 Homework

(29) Prove (51) (hint: let q be arbitrary and consider the quadratic function of t defined by $\phi(t) := \|f - L_n^S f + tq\|_2^2$; use (52)).

(30) page 218, number 6

(31) page 219, number 7

6 Differences, more interpolation, Chapter 6

We are skipping this, except for the parts already described relating to Lagrange interpolation.

- Chapter 6, Section 3.1: Remainder term for Chebyshev and equidistant interpolation,
- Chapter 6, Section 3.4: Divergence of equidistant interpolation

7 Numerical integration, Chapter 7

Quadrature is a way to approximate integrals.

7.1 Interpolatory quadrature, Chapter 7, section 1

The idea behind interpolatory quadrature is to define the approximate integral as the integral of an interpolant (or approximant):

$$Qf = \int_a^b Lf(x) dx = \sum_{i=0}^n f(x_i) \int_a^b \phi_i(x) dx \quad (55)$$

where L is one of the operators we constructed: Lagrange (you choose the points), Bernstein (unusual choice), piecewise-polynomial interpolant (this is called a composite rule). Least-squares approximation does not lead to such a quadrature rule because it is defined in terms of an integral. The error is easy to estimate:

$$Qf - \int_a^b f(x) dx = \int_a^b Lf(x) - f(x) dx \quad (56)$$

7.2 Newton-Cotes, Chapter 7, section 1.1

Here we have Lagrange interpolation with equally spaced points. It is of interest to note that the coefficients

$$\alpha_i := \int_a^b \phi_i(x) dx \quad (57)$$

are positive or not. For the open N-C, they are not. When the coefficients are negative, bad things can happen.

7.2.1 Homework

(32) Prove that Simpson's rule is exact for cubics (hint: use symmetry).

7.3 Gaussian quadrature, Chapter 7, section 3

Gaussian quadrature may be defined by taking the points x_i such that we get a formula exact for as high a degree as possible. Stated as a system of equations, it is highly nonlinear. With n values of x 's and n values of α 's, we might expect to integrate a polynomial of degree $2n - 1$ exactly. Fortunately, if we take the x_i 's to be the roots of the orthogonal polynomial P_n , all is well. First of all, we know the roots are in the interval in question and that they are distinct.

Suppose that $f \in \mathcal{P}_{2n-1}$. Then $f - L_n f$ vanishes at the roots of P_n , so we can write $f - L_n f = P_n q$ where $q \in \mathcal{P}_{n-1}$.

$$Qf - \int_a^b f(x) dx = \int_a^b Lf(x) - f(x) dx = \int_a^b P_n(x)q(x) dx = 0 \quad (58)$$

because P_n is orthogonal to \mathcal{P}_{n-1} .

Fortunately, the coefficients α_i are also positive. Let $f(x) = P_n(x)^2/(x - x_i)^2$. By (58), since the degree of f is $2n - 2$,

$$\alpha_i f(x_i) = Qf = \int_a^b f(x) dx > 0 \quad (59)$$

since f is positive except at the x_i 's. We also have $f(x_i) > 0$ since P_n has only a simple zero there.

7.3.1 Homework

(33) Determine the Gauss points x_i^n for $i = 1, \dots, n$ for the unit interval for $n = 1, 2, 3$. (Hint: use symmetry as much as possible.)

8 Numerical solution of o.d.e's

The simplest differential equation to solve is an ordinary differential equation

$$\frac{du}{dt} = f(u, t) \quad (60)$$

with initial value

$$u(0) = u_0 \quad (61)$$

where we are interested in solving on some interval $[0, T]$.

The definition of the derivative as a limit of difference quotients suggests a method of discretization:

$$\frac{du}{dt}(t) \approx \frac{u(t + \Delta t) - u(t)}{\Delta t} \quad (62)$$

where Δt is a small positive parameter. This suggests algorithms for generating a sequence of values $u_n \approx u(n\Delta t)$ given by (for example)

$$u_n = u_{n-1} + \Delta t f(u_{n-1}, t_{n-1}) \quad (63)$$

or by

$$u_n = u_{n-1} + \Delta t f(u_n, t_n) \quad (64)$$

where $t_n = n\Delta t$.

The algorithm (63) is called the **explicit Euler** method, and the algorithm (64) is called the **implicit Euler** method. It can be shown that both generate a sequence with the property that

$$|u(t_n) - u_n| \leq C_{f,T} \Delta t \quad \forall t_n \leq T \quad (65)$$

provided that we solve the implicit equation (64) for u_n exactly and we compute with exact arithmetic. This is proved in Theorem 1 of section 8.1 on page 368.

The issue of solving the nonlinear equation in the implicit Euler method (64) at each step is important but not a show-stopper; one uses the methods we have studied for solving nonlinear equations. Moreover, we are in a situation in which we have a very accurate approximation to the solution, e.g., one given by the explicit Euler method (63).

However, the requirement of using finite-precision arithmetic means that the best error behavior we could expect is

$$|u(t_n) - u_n| \leq C_{f,T} \Delta t + n\epsilon \quad \forall t_n \leq T \quad (66)$$

where ϵ measures the precision error that occurs at each step in (64). It is useful to re-write (66) using the fact that $n = t_n/\Delta t$ as

$$|u(t_n) - u_n| \leq C_{f,T} \Delta t + \frac{t_n \epsilon}{\Delta t} \quad (67)$$

which shows that the error reaches a minimum and cannot be reduced by reducing Δt . This is illustrated well in Figure 1 on page 376.

The first way presented in the book to deal with this is surely the simplest: centered differences. This provides a stable second-order approximation. However, it does not generalize very well. In section 1.4 of chapter 8, the book derives what might be called a forward differentiation formula

$$\frac{du}{dt}(t) \approx P'(t_{n-1}) = \frac{1}{\Delta t} \sum_{i=0}^k a_n^k u_{n-i} = f(u_{n-1}, t_{n-1}). \quad (68)$$

where P is a polynomial interpolating the values u_j at t_j . The coefficients a_n^k are just the derivatives of the basis functions for this interpolation process. If we choose P to be of degree k , we have chosen to provide maximum accuracy Δt^k . For $k = 1$ this is explicit Euler, and for $k = 2$ this is the centered difference scheme. However, for $k = 3$ the scheme is unconditionally unstable. We find solutions that grow like ξ^n where

$$-\xi = \frac{5 + \sqrt{29}}{4} \approx 2.5963 \quad (69)$$

even if we have $f = 0$.

Another way to increase the accuracy in (66) is to use backwards differentiation formulae (BDF)

$$\frac{du}{dt}(t) \approx \frac{1}{\Delta t} \sum_{i=0}^k a_n u_{n-i} = f(u_n, t_n) \quad (70)$$

where the coefficients $\{a_i : i = 0, \dots, k\}$ are chosen so that (70) is exact for polynomials of degree k . The BDF for $k = 1$ is the same as implicit Euler. Using the approximation (70), we get an algorithm of the form

$$\sum_{i=0}^k a_i u_{n-i} = \Delta t f(u_n, t_n) \quad (71)$$

which can be solved for u_n provided $a_0 \neq 0$. In this case, the final error estimate would be

$$|u(t_n) - u_n| \leq C_{f,T,k} \Delta t^k + \frac{t_n \epsilon}{\Delta t}. \quad (72)$$

Ultimate accuracy is still limited, but smaller absolute errors (with larger Δt) can be achieved with higher values of k . For example, suppose that

- $\epsilon = 10^{-6}$ (which corresponds to single precision on a 32-bit machine)
- $T = 1$ and
- (for the sake of argument) $C_{f,T,k} = 1$.

Then with implicit Euler ($k = 1$) the smallest error we can get is 10^{-3} with $\Delta t = 10^{-3}$. On the other hand, with $k = 2$ we get an error of size 10^{-4} with $\Delta t = 10^{-2}$. Not only is this a smaller error but less work needs to be done to achieve it. In practice, the constant $C_{f,T,k}$ would depend on k and the exact error behavior would likely be different in detail, but the general conclusion that a higher-order scheme may be better still holds. The BDF methods for $k = 2$ and 3 are extremely popular schemes.

We see that higher-order schemes can lead to more manageable errors and potentially less work for the same level of accuracy. Thus it seems natural to ask whether there are limits to choosing the order to be arbitrarily high. Unfortunately, not all of the BDF schemes are viable. Beyond degree six, they become **unstable**. Let us examine the question of stability via a simple experiment. Suppose that, after some time T_0 , it happens that $f(u, t) = 0$ for $t \geq T_0$. Then the solution u remains constant after T_0 , since $\frac{du}{dt} \equiv 0$. What happens in the algorithm (71) is that we have

$$\sum_{i=0}^k a_i u_{n-i} = 0 \quad (73)$$

for $n \geq T_0/\Delta t$. However, this does not necessarily imply that u_n would tend to a constant. Let us examine what the solutions of (73) could look like.

Consider the sequence $u_n := \xi^{-n}$ for some number ξ . Plugging into (73) we find

$$0 = \sum_{i=0}^k a_i \xi^{-n+i} = \xi^{-n} \sum_{i=0}^k a_i \xi^i \quad (74)$$

If we define the polynomial p_k by

$$p_k(\xi) = \sum_{i=0}^k a_i \xi^i \quad (75)$$

we see that we have a null solution to (73) if and only if ξ is a root of p_k . If there is a root ξ of p_k where $|\xi| < 1$ then we get solutions to (73) which grow like

$$u_n = \xi^{-n} = \left(\frac{1}{\xi}\right)^{t_n/\Delta t}. \quad (76)$$

Not only does this blow up exponentially, the exponential rate goes to infinity as $\Delta t \rightarrow 0$. This clearly spells disaster. On the other hand, if $|\xi| > 1$, then the solution (76) goes rapidly to zero, and more rapidly as $\Delta t \rightarrow 0$. For roots ξ with $|\xi| = 1$ the situation is more complicated, and $\xi = 1$ is always a root because the sum of the coefficients a_i is always zero. Instability occurs if there is a multiple root on the unit circle $|\xi| = 1$. In general, one must consider all complex (as well as real) roots ξ .

Now that we know what to look for, let us return to the question of the higher-order BDF methods. It is well known (see equation 11 on page 284) that

$$\sum_{i=0}^k a_i u_{n-i} = \sum_{j=1}^k \frac{(-1)^j}{j} \Delta^j u_n \quad (77)$$

where we define Δu_n to be the sequence whose n -th entry is $u_n - u_{n-1}$. The higher powers are defined by induction: $\Delta^{j+1} u_n = \Delta(\Delta^j u_n)$. (There is multiple notation abuse here: Δu_n really means the n -th element of the sequence $\Delta\{u.\}$ where $\{u.\}$ denotes the full sequence.) For example, $\Delta^2 u_n = u_n - 2u_{n-1} + u_{n-2}$, and in general Δ^j has coefficients given from Pascal's triangle. We thus see that $a_0 \neq 0$ for all $k \geq 1$.

Given this simple definition of the general case of BDF, it is hard to imagine what could go wrong regarding stability. Unfortunately, the condition that $|\xi| \geq 1$ for roots of $p_k(\xi) = 0$ restricts k to be six or less for the BDF formulæ. Presumably, the same feature that makes the forward difference formula fail at $k = 3$ is at work. We simply cannot compute such accurate approximations to the derivative by looking so exclusively in one direction in a stable way.

8.0.2 Homework

- (34) Determine the characteristic polynomial for the centered difference scheme and find its roots. Are then inside or outside the unit circle?
- (35) Determine the characteristic polynomial for the second-order backwards difference method and find its roots. Are then inside or outside the unit circle?

8.1 Multi-step methods, Chapter 8, section 2

The concept of multi-step methods is to improve stability by utilizing previous f values. The metaphor is to approximate the integral of f instead of the derivative of u . They can be both implicit and explicit. The predictor-corrector concept involves a mixture of the two.

8.1.1 Homework

- (36) page 394, number 1.

8.2 One-step methods, Chapter 8, section 3

Runge-Kutta schemes approximate the integral over a single interval by what might be viewed as a complex predictor-corrector scheme for internal values that will be discarded at the end of the step.

8.3 Linear difference equations, Chapter 8, section 4

This section develops the general stability theory for difference methods.

8.4 Consistency, convergenc and stability, Chapter 8, section 5

This section shows that stability and consistency (accuracy) are equivalent to convergence.

References

- [1] A. Berman and R. J. Plemmons. *Nonnegative matrices in the mathematical sciences*. Academic Press, 1979.