Scheid Numerical Analysis Book Notes **(Francis Scheid, 1968)** PhL started 10.13.04
(in the Schaum's Outline Series) finished 12.3.04

Scheid's approach for each chapter is to have an introductory section which summarizes the main results of the chapter all in one place, perhaps including key equations, and then in the "worked problems", he develops the material summarized up front. In a "normal text", those worked problems would be in the main line text. Due to this method, which I like a lot in fact, sometimes you cannot really understand what the summary section is saying because you don't yet know enough! So if you want to "read a chapter of Scheid", you often really need to read through the worked problems.

Scheid does not have a detailed table of contents, so I have made one in a separate document. This also gives a summary of the main subjects in each chapter. There is so very much stuff!

****

Chapter 1: What is Numerical Analysis **p 1**

The main idea here is that there is usually error in both the input data and in "The Algorithm" which creates the output data.

****

Chapter 2: The Collocation Polynomial **p 10**

Let's now look at the claims:

Let y(x) be some arbitrary function, and imagine that there is some poly p(x) of degree n that exactly matches y(x) at a set of n+1 discrete points x_0 thru x_n.

(1) The claim is that such a poly p(x) does in fact **exist**, and that there is **only one** such p(x)! Proof of this occurs in some later chapter! This poly p(x) is called **"THE collocation polynomial for y(x) of degree n"**. Example: n+1 = 2 points, ∃ one poly of degree n= 1 (straight line) that "fits" these two points.

Comments: in later chapters, we will first find what p(x) is for equally spaced arguments, then in a still later chapter we will see what it is for unequally spaced arguments.

(2) If p(x) is ANY poly of degree n, and if r is ANY number, you can find a unique poly q(x) and R such that p(x) = (x-r) q(x) + R, and q(x) will be of degree n-1. This is proven in problem 2.1. This is called **the division algorithm**, and you can compute q(x) quickly using synthetic division as in problem 2.3.

(3) Obviously, based on the above, p(r) = R, which is called **the remainder theorem**.

(4) If p(x) is ANY poly of degree n, and if r is ANY number, and if p(r) = 0, then (x-r) is a factor of p(x). This is called **the factor theorem**. I guess given p(x) you could use (2) above to get p(x) = (x-r) q(x) + R, and then you conclude that if p(r) = 0, you must have R = 0, so then p(x) = (x-r) q(x) and QED.

Comments: The above says that if a poly has p(r) = 0, then it must be possible to "factor out" (x-r) from p(x). I guess this seems fairly obvious since I know you can fully factor any polynomial, even if roots are complex, but here we are in the space of reals.

(5) If p(x) is ANY poly of degree n, it can have **at most n zeros**. Of course if it has n zeros, then it has n roots as noted above and you can then write p(x) = product of $(x-r_i)$. If it had n+1 or more zeros, then that product form would still result, but then you have a poly of degree n+1 or more, which is a contradiction.

Comments: The poly can have *less than* n real zeros if, when you factor it, some roots are complex. For example, take $p(x) = x^2 + 1$, a parabola dipping down to +1 at x=0. Obviously no real zeros. Roots are $\pm i$.

(7) Define $\pi(x)$ as the product of n+1 factors $(x - x_i)$. Obviously $\pi(x)$ is a poly of degree n+1 and has zeros at the n+1 points shown. Let y(x) be an ARBITRARY function, and let p(x) be the official collocation polynomial of degree n for y(x). Then the following claim is made:

$$y(x) - p(x) = \pi(x) * y^{(n+1)}(\zeta) / (n+1)! \qquad \text{where } \zeta \text{ is some value in the range } (x_0, x_n)$$

Note that this form does show the "collocation" idea in that the difference is zero at each of the n+1 collocation points. All we know is that ζ exists, but in general we don't know what it is -- and it is a function of x. This formula can be used to see how far y(x) can wander away from p(x) as you move between collocation points. An example is given in problem 2.9 of using this formula to put a bound in the difference y(x) - p(x) even when you have no idea what ζ is.

Comments: We still don't know how to construct p(x) at this point, but the above item is talking about how you might measure the *error* in your fit using p(x).

Problem 2.7 Revisited. *Ignore* the first equation and define F(x) as shown where C = constant. [The reason to ignore the first equation is that it makes you think F(x) ≡ 0 later on.] Now F(x) really is some non-zero function. It is then true that F(x) = 0 at the n+1 collocation points. We then pick an arbitrary but fixed point and call it x_{n+1} and we then define C as shown (C is then a function of x_{n+1}). Now we do then know that F(x) has at least n+2 zeros -- the original n+1 collocation points, plus our new point x_{n+1}. We then apply **Rolle's Theorem** a bunch of times and conclude that $F^{(n+1)}(x)$ has at least one zero and we call that point ζ, and of course everything is a function of our x_{n+1}. NOW we differentiate both sides of our original F(x) equation wrt x n+1 times, and only the leading term in $\pi(x)$ survives and we get the result shown which is therefore another way to write C. We then combine this C equation with the previous one to get the next one, where $\zeta = \zeta(x_{n+1})$. Then we note that x_{n+1}= an arbitrary point, so the equation must be true at any x and we are done. [As for **Rolle**, if a function has n+2 zeros, then between each adjacent pair of zeros Rolle says there is a place where slope = 0, so that means that F'(x) will have n+1 zeros etc.]

****

Chapter 3: Finite Differences p 15

Instead of having a function y(x), we have a set of sample points y_k. The first difference Δy_k is like a first derivative or slope, the second $\Delta^2 y_k$ is like the second derivative or curvature, and so on. In calculus, you might write $d^n y_k(x)/dx^n$ as the nth derivative, but that is all you normally have to say. You could write this as an elaborate limit. In differences, $\Delta^n y_k$ (the nth difference) can be related explicitly to the neighboring n sample values by the formula on page 15 which contains the binomial coefficient.

The chapter goes on to state the "rules" for differences. Many are like those for differentiation, such as rules 1,2,3,4,5. In the last two, however, there is a subtle subscript detail that is critical. Rule 7 for a power is completely new, although trivial to derive. The calculus thing would be $d/dk(C^k) = \ln C * C^k$, so it is pretty vague how $\ln C \rightarrow (C-1)$ in the discrete limit.

Notice that you always "go to the right", not the left. For example. $\Delta y_k = y_{k+1} - y_k$. [These would be called a set of "**forward**" differences.]

Rule 8 and 9 for sines, cosines and logs are very new to me. In Rule 9, unlike all other rules, author assumes equally spaced samples, spacing is h. You get the vague idea of $(\log(x))' = 1/x$, but it is "remote", as Scheid remarks.

Rule 10 gives the "delta function", but is called "the error function". One isolated unit $y_k = 1$.

Rule 11 is the derivative of a delta function, 0000000... -1 +1 0000000

****

Chapter 4: Factorial Polynomials p 22

Well, let's look at one example: $y_k = k^{(3)} = k(k-1)(k-2) = k^3 - 3k^2 + 2k = k! / (k-3)!$

This is a "polynomial" not in variable x, but in the index k, which index labels the samples. The case n=3 above shows that we have a "polynomial" of degree 3. When you expand the factorial form into the series of power terms, the coefficients are NOT the binomial ones, but are instead the **Stirling first kind**, see page 22. You can do the reverse and expand a single power in a set of factorial polynomials, and then you have the **Stirling second kind** coefficients, as on page 23.

Why are these factorial things interesting? For one thing, consider: $\Delta k^{(n)} = n\, k^{(n-1)}$
Although $k^{(n)}$ is not the power k^n , it behaves that way wrt differentiation!
So maybe these $k^{(n)}$ are the better discrete analogs of the continuous-argument powers.

Also, as we well know, $k^{(n)}$ appears in the binomial coefficient as shown page 22.

We have implicitly proven a theorem: the difference of a poly of degree n is one of degree n-1. This is pretty obvious just from doing a power: $Dk^3 = (k+1)^3 - k^3$ so the leading k^3 terms cancel. But the result is even more obvious in terms of the $k^{(n)}$.

If you want the difference of a poly in powers of k, one way is to express it first in terms of $k^{(n)}$ and then use the simple difference rule shown above.

****

Chapter 5: Summation **p 30**

(1) Suppose you want to sum a series z_k. *If* you can find y_k such that $\Delta y_k = z_k$, then the sum is trivially given by $y_n - y_0$. This is like saying $\int dy = y_2 - y_1$. In this case, you have a "**telescoping** sum". This term arises from the idea of a collapsing telescope which might have 5 sections. The extended telescope is analogous to the full series you want to sum up. But terms cancel pairwise, and you end up only with the first and last terms. This is analogous to compressing the telescope down to one little section.

(2) The idea of doing "summation by parts", analogous to "parts integration".

****

Chapter 6: The Newton Formula **p 34** // for the equi-spaced collocation poly

On page 15 in Chapter 3 we expressed $\Delta^k y_0$ as a sum of the y_k values with binomial coeffs. Here Scheid will derive the inverse formula which gives y_k as a summation over the $\Delta^k y_0$. Again, there is a binomial coefficient. The sum could not be simpler!

$$y_k = \sum_{i=0}^{k} \binom{k}{i} \Delta^i y_0 \quad = \sum_{i=0}^{k} [\, k^{(i)}/i! \,]\ \Delta^i y_0 \qquad \text{// inverse of page 15}$$

Notice that the right side has k+1 terms, and that each term is a factorial poly in k, so the entire right side is a polynomial (of degree k) in the index k. At each value of k, this poly matches your value set y_k .

Now consider instead this sequence of terms:

$$p_k = \sum_{i=0}^{n} \binom{k}{i} \Delta^i y_0$$

where the summation now runs to some positive integer $n \geq k$. Suppose n = 4. For k=0, the sum on the right appears to have 5 terms, but really there is only 1 term, the one with i = 0 because (k,i) = 0 when i>k as you can show from the recursion formula for binomials. So $p_0 = y_0$. For k = 1, there are really only 2 terms, and so on. For k = 4, there really are five terms, and we still have agreement $p_4 = y_4$. The only claim made about the above formula with the sum to n is that it agrees with the y_k for the first n+1 values in the sequence. We don't care about higher points like y_5. We care about $k \leq n$. So for n = 8, we get a poly of degree 8 in k which matches the values y_k at the 9 points y_0 through y_7.

[**Comment Added**: In the first formula above for y_k, we cannot "continue " k off the integers because k appears as a sum end point. But the sum for p_k above CAN be so continued. Since the p_k sum agrees with the y_k sum on the integers, we can regard p_k as a reasonable continuation to continuous k. Therefore, the formula for p_k can be thought of as p(k) with x = k being a continuous variable, and what we then have is a polynomial p(k) which "fits" our n+1 data points y_0 , y_1, y_2 ... . In Chapter 2 Scheid talked about the existence of a collocation polynomial, and here finally we have its exact construction. This collocation polynomial fits data samples which are equally spaced by 1 unit and start at 0. We can of

course define an x variable $x = x_o + hk$ to allow for arbitrary but equal point spacing, and then our values of x corresponding to the integers in k are $x_k = x_o + hk$. Then we can say:

$$P(x) = p(k) = \sum_{i=0}^{n} \binom{k(x)}{i} \Delta^i y_0 \qquad \text{where } k(x) = (x - x_0)/h$$

and this then gives you the collocation polynomial that matches (x_0, y_0) , (x_1, y_1) ... (x_n, y_n). In Chapter 2 we showed that the P(x) is unique and of course here we see that it "exists". In Chapter 8, we will see how we can generalize to get P(x) for unequally spaced arguments.]

Problem 6.7 gives an example. We are interested in n = 4. We have 5 coordinates of the form (x_k, y_k) and these are shown in the table. It happens that the x_k arguments are spaced by 1, so we can use the above formula for the p_k . The result is shown. This result p_k is a polynomial in k, and this polynomial runs through all 5 points. You could even consider k to be a real variable and then p(k) hits all the discrete points of your model.

The formula above for p_k is called **Newton's Formula for the Collocation Polynomial of order n**. This would be useful if you need to "interpolate" some data between equally spaced samples, such as a table of tangents, or some experimental data.

In the case that the x_k are spaced by some number h other than 1, you can write p_k as $p(x_k)$. The idea is the same, you get a polynomial in x_k that hits the first n+1 points y_k of your data, see problem 6.5.

So up to this point we are always talking equally-spaced x_k values.

****

Chapter 7: Operators and Collocation Polynomials p 38

We already know about operator Δ. They define E^x such that $E^x\, y_k = y_{k+x}$. Normally E just moves you ahead to the next data sample, BUT they allow sometimes for x = 1/2, meaning you move just a half sample ahead. The operator ∇ is the difference going to the left, $y_k - y_{k-1}$.

Now finally we come to operators δ and μ. You can see that δ is a better "derivative" since it is centered at the point of your interest:

$$\delta\, y_k = y_{k+1/2} - y_{k-1/2} \qquad \mu\, y_k = (y_{k+1/2} + y_{k-1/2}) / 2$$

And similarly, operator μ is the average shown on the right above.

Using the operators μ and δ gives us various alternate forms for the Newton collocation polynomial, and each form has a name: **Gauss forward**, **Gauss backward**, **Stirling**, **Everett**, **Bessel**. At this point in my reading, I have no idea why any of these formulas is better than another (but see comments below). Still, I think I could derive any formula. The notion of operators is very familiar to me, so that part is easy.

It is important to realize that there is really only ONE collocation polynomial that fits n+1 points. Each of these methods causes the terms of that polynomial to be ordered in a different manner. Since we deal with finite order n, rearrangement is not an issue.

Comments: I have now looked in more detail at this chapter (11/20/05). First of all, I now see how the notion of "operators" is very useful in proving formulas which otherwise might be messy to prove.

This is similar to things like Campbell-Hausdorf and such, I am much at home with operators as tools. Everything makes sense.

All the various alternative formulas are derived in the worked problems.

The most fascinating result perhaps is the idea on page 50 that for each "zig zag path" going through the δ difference tree, you really have "A Formula". Each path corresponds to a particular ordering of the terms. I did not prove this, but it seems reasonable. Then the last problem shows how all the various formulas of this chapter can be mapped to zigzags. At once you see the weakness of the original Newton formula because your path through the data tree moves a large distance vertically, so you wander away from the value of y_k that you are most interested in. In contrast, the formulas which use δ (which is to say, ALL the other formulas) keep you at a more or less constant horizontal level, so you DON'T wander away from your y_k of interest.

Remember that we are using these formulas in the first place to construct a fitting polynomial in the region of some argument k, so it seems logical that we would prefer not to incorporate too much difference data that comes far away from k, where the function y(x) that "made" the data might be significantly different. It is true that if y(x) is itself a polynomial, then all the "formulas" (all zigzag paths) give the same result IF we follow the zigzag all the way to the end of the tree. But the point I think is that you might only want to go 2, 3 or 4 levels of the tree to get a polynomial of degree 2,3 or 4 for your fit to the n+1 points, and in this case, different paths give different polynomials due to the ordering difference. It is here that the δ formulas are surely better than the original Newton formula.

****

Chapter 8: Unequally-Spaced Arguments. p 53

Here we write *an exact formula* for the collocation poly p(x) of degree n which passes through some arbitrary set of n+1 data points (x_k, y_k), even if the x_k are unequally spaced. Here is that formula, which is known as **Lagrange's Formula for the collocation poly**,

$$p(x) = \sum_{i=0}^{n} L_i(x)\, y_i \qquad \text{where } L_i(x) = F_i(x) / F_i(x_i) \;=\; \text{"a Lagrange multiplier"}$$

where $F_i(x)$ is the product π(x) but with the one factor $(x-x_i)$ missing. That is

$F_i(x) = \pi(x) / (x - x_i)$ // of degree n, since π(x) was degree n+1

The key fact to notice is that $L_i(x)$ is in general a poly of degree n in x, but that $L_i(x_k) = \delta_{ik}$. In other words, for example, $L_3(x_5) = 0$ because the numerator vanishes from factor $(x - x_5)$, but $L_5(x_5) = 1$ because the $(x - x_5)$ is now missing from the numerator, and the num and den factors are now identical.

$L_3(x_5) = F_3(x_5) / F_3(x_3) = 0 / \text{stuff} = 0$
$L_5(x_5) = F_5(x_5) / F_5(x_5) = a/a = 1$

This Kronecker fact makes a proof of the above summation formula trivial.

$p(x_j) = \sum_{i=0}^{n} L_i(x_j)\, y_i = \Sigma\, \delta_{ij} y_i = y_j$ so we pass through all the points.

Again, all the $L_i(x)$ are of degree n, so of course p(x) is of degree n.

Remember that the collocation polynomial is unique, so this is the only one (chapter 2).

A **determinant form** is given for writing p(x) on page 53. The proof that this is correct is trivial. First, it is a poly on degree n, and second, it is true at each of the n+1 points because two rows are the same. So no need to relate this to the Lagrange form directly. Uniqueness tells us it is the same. Scheid points out that although nice, it is not very useful. (Reminds me of recurrence relation stuff in Bateman.)

The **Aitken's Method** gives you a graphical way similar to a difference tree to "build up" the collocation polynomial given some data. In this method (starts in problem 8.6), you only do 2x2 determinants to compute each item in the tree. For each column in the tree, the data elements are functions of what is in the last column on the right, and the one column before. So yes, can do Excel to do this.

Note 1 Added. We can compute that

$$\begin{aligned}\pi'(x) &= 1*\pi(x)/(x-x_0) + 1 * \pi(x)/(x-x_1) + \text{etc} \\ &= \text{deriv of first term times rest + deriv of second term times rest + ...} \\ &= \pi(x)\,[\ 1/(x-x_0) + 1/(x-x_1) + 1/(x-x_2) + ... \\ &= \prod_{k \neq 0} (x - x_k) + \prod_{k \neq 1} (x - x_k) + \prod_{k \neq 2} (x - x_k) + ...\end{aligned}$$

Therefore, we see that, for example, $\pi'(x_2) = \prod_{k \neq 2} (x_2 - x_k)$ because only that one term is non zero in the sum. More generally, we can claim that $\pi'(x_i) = \prod_{k \neq i} (x_i - x_k)$. This is *exactly* the denominator of $L_i(x)$.

Note 2 Added: Why does $L_i(x)$ work? Because the term in the sum marked by i has y_i as its amplitude. Since $L_i(x_j) = \delta_{i,j}$, we see that $p(x_i) = y_i$ at each i. Therefore, this p(x) does for sure match all y_i and it is for sure a poly of degree n+1, and we know that only one such poly can match, so this is it!

****

Chapter 9: Divided Differences p 58

Divided differences are the *generalization* of ordinary (equal spaced) differences to the situation of unequal spacing of the arguments x_k. Recall that in the case of equal spacing, we found this formula for our collocation polynomial (Newton's formula, Chapter 6),

$$p_k = \sum_{i=0}^{n} \binom{k}{i} \Delta^i y_0 \quad \text{OR:} \quad P(x) = p(k) = \sum_{i=0}^{n} \binom{k(x)}{i} \Delta^i y_0 \quad \text{where } k(x) = (x - x_0)/h$$

Let's write out the first few terms of this series which in full goes up to some integer n: (remember that points are still equally spaced here)

$$P^2(x) = y_0 + [(x - x_0)/h]\,\Delta^1 y_0 + (1/2)\,[(x - x_0)/h]\,[(x - x_0 - h)/h]\Delta^2 y_0$$

$$= y_0 + (x - x_0)\,\{\Delta^1 y_0/h\} + (x - x_0)(x - x_1)\,\{\Delta^2 y_0/2h^2\} + (x - x_0)(x - x_1)(x - x_2)\{\Delta^3 y_0/3!\,h^3\} + \ldots$$

In this chapter, this result is generalized to become **Newton's Divided Difference collocation formula**, which looks similar to the above, but is this:

$$p(x) = y_0 + (x - x_0)\,\{y(x_0,x_1)\} + (x - x_0)(x - x_1)\,\{y(x_0,x_1,x_2)\} + (x - x_0)(x - x_1)(x - x_2)\{y(x_0,x_1,x_2,x_3)\} + \ldots$$

The coefficient labeling is a little confusing. Rather than $y(x_0,x_1,x_2)$, we really should have $k(x_0,x_1,x_2;\, y_0,y_1,y_2) = k(\mathbf{r_0}, \mathbf{r_1}, \mathbf{r_2})$. These coefficients are functions of the three sets of coordinates. These coefficients are the "divided differences" that this chapter is about. Note that this is an alternative approach to the unequal spacing Lagrange collocation formula.

At the top of page 59, we see how the nth divided difference telescopes down, if you will, do the simple n^{th} difference if spacing is equal. In this case, the above two formulas are the same.

So what are these "divided differences". The first one is fine, but the higher order ones are increasingly messy to write out in terms of the basic y_k and x_k. However, it is easy to write them out in a stepwise fashion from the previous one, and the leads to the simple tree representation shown, which we also use for regular differences, except here we always have an extra denominator thing to worry about.

In the problems, Scheid shows that each divided difference is symmetric in its arguments, which makes life a little simpler. The actual formula is derived in problem 9.8.

Now recall from Chapter 1 we proved this result,

$$y(x) - p(x) = \pi(x) * [\, y^{(n+1)}(\zeta) / (n+1)! \,] \qquad \text{where } \zeta(x) \text{ is some value in the range } (x_0, x_n)$$

It turns out that the factor in square brackets is in fact the following divided difference:

$$y^{(n+1)}(\zeta) / (n+1)! = y(x,x_0,x_1,x_2 \ldots x_n)$$

Notice that this has that extra first argument x, so this verifies our early claim that $\zeta = \zeta(x)$. This error factor is a function of all n+1 point coordinates (x_k, y_k), and is also a function of $y = y(x)$ and x. Obviously y(x) has to be in there somewhere, since y(x) can be an arbitrary function, and this is a formula for the exact error from the polynomial fit. I guess it is interesting that you can write the error in this way, but doubt it is very useful!

You might ask: how does this Newton Divided Difference formula compare the Lagrange formula for what must be the exact same collocation polynomial? It must be some sort of rearrangement. Some notion of the connection is provided by the result on page 58 which gives this formula for the divided differences:

$$y(x_0,x_1,x_2 \ldots x_n) = \sum_{i=0}^{n} y_i \;/\; F_i(x_i)$$

where $F_i(x)$ is the product of all the $(x - x_\ell)$ factors $\ell = 0$ through n *except* the factor $(x - x_i)$, $\ell = i$. Good thing since in $F_i(x_i)$ that factor would be $(x_i - x_i) = 0$. Notice that $F_i(x)$ includes the factor $(x - x_j)$ for any j that is different from i, and that therefore $F_i(x_j) = 0$ since it contains factor $(x_j - x_j)$.

For comparison, here is the Lagrange formula again,

$$p(x) = \sum_{i=0}^{n} y_i \; [\, F_i(x) / F_i(x_i) \,] \qquad \text{// note that } [\, F_i(x_j) / F_i(x_i) \,] = \delta_{i,j}$$

which we compare with the divided difference Newton formula

$$p(x) = y_0 + \sum_{i=1}^{n} y(x_0,x_1,x_2 \ldots x_i) \; \pi_i(x)$$

where I have defined π_i to be $(x - x_0)(x - x_1)...(x - x_i)$.

In the Lagrange formula, all terms have the same "complexity", and if you evaluate p(x) at some x_k, exactly one of these complex terms survives such and gives the result. In the Newton formula, the terms start simple and get more and more complex. If you evaluate p(x) at x_0, due to $(x - x_0)$, only the first term survives. At x_1 the first two terms survive, and so on. at $x = x_n$ you need ALL the terms.

So as noted already, somehow these two formulas are rearrangements of each other. The first formula is sorted by the y_i values, one for each term. The second formula is sorted instead by ever-increasing products of roots.

****

Chapter 10: Osculating Polynomials p 65

In Chapter 8, we are dealing with a set of n+1 points x_i and we come up with a poly p(x) *of degree n* which matches y(x) at all these points, and this poly is called "the collocation poly". In chapter 2 we came up with a formula that can be used to estimate the difference y(x) - p(x).

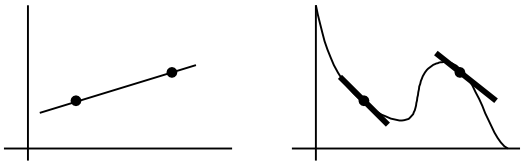
Here, we still use a set of n+1 points x_i , but now we try to find some p(x) *of degree 2n+1* which matches the values AND matches the first derivatives at all the x_i . Obviously, this is going to be a better fit due to the smooth touches, and -- no surprise -- we need a higher order poly to make this work. As in the collocation case, there is an exact formula for the solution, which is called "the osculating polynomial", and that formula is called **Hermite's Formula** on page 65. The U and V functions shown are functions of the Lagrange multiplier, and you can see that U and V are each of degree 2n+1 so p(x) must also be of that degree. A proof of the Hermite Formula is given in problems 10.1 and 10.2 (which I reviewed in detail), and an example of an osculating poly is shown in problem 10.3. Note that this Hermite Formula has nothing whatsoever to do with Hermite Polynomials.

The idea can be generalized to higher order where you match values, slopes and curvatures, say, and probably you need a 3n+1? degree poly to do that. I don't think we will ever need this higher order stuff.

Similar to the simpler collocation case, there is a formula which measures the error y(x) - p(x) and it is shown on page 65. Remember that our starting point was to assume that p(x) matches y(x) at n+1

locations for both value and slope, and we end up with (n+1)*2 = (2n+2) in the final formula. If you started off saying that the p(x) matched only at n points (perhaps called x_1 through x_n), then that error formula would have 2n in place of 2n+2. This is sometimes called "deleting a zero", and you can see that there is nothing to it. The proof of this error formula on page 65 is given in problem 10.4, and it is really identical to the proof of the corresponding collocation formula!

Note Added: All we learn here is that you can fit a set of n+1 points *and* n+1 derivatives with a poly of degree 2n+1, and we have the explicit formula that does it (Hermite's Formula). Earlier we were able to fit a set of n+1 points with a poly of degree n, so the additional requirement of fitting the slopes as well has cost us an extra n+1 in degree. As an example with n= 1 consider, (there are n+1 = 2 points)



On the left we fit the two points with a degree 1 poly, but on the right, given in addition the initial slope requirements shown as dark segments, you can see that a degree 2 poly cannot do the job. It turns out that degree 3 can do the job, and that is 2n+1.

****

Chapter 11: The Taylor Polynomial p 70

If a function is analytic at x_0, you can do an infinite Taylor series expansion around x_0. This Taylor thing which I have used all my life is of course a polynomial if you stop somewhere, and Scheid describes it as the ultimate in osculation, since it matches derivatives up to whatever order you allow the Taylor series to be, but only at a single point shown as x_0.

As usual, the meat is in the worked problems. The Taylor series is used as a "theme" to teach us many things we have never heard of before. As usual, Scheid has engineered the sequence of his problems in precision order, where each one provides just what is needed in the next one. Sort of just in time delivery.

The topics are:

- the **Taylor formula** and applying it ; definition of analyticity
- the **error formula** and applying it to see how many terms you have to add up
- introduction of operator D and relations between **D and Δ**
- introduction of operators $\mathbf{D^{-1}}$ **and** $\mathbf{\Delta^{-1}}$ and relations between them
- the **Euler Transformation** for re-ordering a series
- the **Bernoulli** numbers
- the **binomial expansion** as a Taylor series, then later applied to operators
- the **Euler-McLaurin** formula for writing a series as integral + corrections.

In 11.1, we **derive the Taylor formula**, easily done. We just expand around $x = x_0$ and find the coefficients.

In 11.2, we use this formula to get the expansion series for e^x.

In 11.3 he derives the ***integral formula*** **for the error R** = y(x) - p(x), where p(x) is a finite Taylor series fit to y(x). Everything depends on x_0 because that is the point we are expanding around. The proof is quite easy.

Then in 11.4, we use that integral formula and the usual mean value theorem to rewrite our error R in a form involving the usual ζ , some unknown point in our interval (x_0, x). But they we discover that this error expression has the exact form of the next Taylor series term, but with x replaced with ζ. This form of the error is called **Lagrange's Error Formula** for a Taylor series fit.

In 11.5 we apply this error formula to doing a fit of e^x. We can then use this formula to decide how many terms we need to use in our Taylor fit to get the error below some level.

In 11.6 we define **D = h d/dx** as an operator.

In 11.7 we just write the Taylor series using D^i to do the derivatives.

In 11.8 he says that one definition of **analyticity** at x_0 is that the Taylor series converges, meaning the limit of the error R is 0. Then we can talk about infinite series.

In 11.9 we can then write the entire infinite series Taylor expansion as : $\mathbf{e^{kD}\ y_0 = y_k}$. Maybe easier first to think of this as $e^{kD}\ y(x_0) = y(x_k) = y(x_0 + hk) = y(x)$. This of course reminds us that D is the generator of translations in my previous life of Lie algebras and Groups. The Taylor series is just a spatial translation operator.

In 11.10, we can translate one unit ahead using this operator, so $\mathbf{e^D = E}$. And $E = \Delta + 1$, so we have now provided a relationship between the finite difference operator Δ and the continuous one D. The connection is that $\mathbf{e^D = 1 + \Delta.}$

In 11.11 we examine the Taylor expansion of ln(1+x) and show that it converges.

In 11.12, we use our Lagrange error formula to find that we need to add up 2000 terms if we want 3 place accuracy, so this is a lousy series!

In 11.13 we solve the above equation relating D and Δ for D: $\mathbf{D = ln(1+\Delta)}$ = the series. Scheid says that we do have to be more series on convergence issues, but the operators tell us the right answers very quickly.

In 11.14 we do $\mathbf{(1+x)^p}$ as a Taylor series and of course we get the binomial thing. He comments in passing that you can think of this for general p, and this means generalizing the binomial coefficient, and of course we know how to do that with gamma functions.

In 11.15 we use $(1+E)^{-1} = 1 - E + E^2$ ... to ponder a sum of terms a_i which have alternating signs. Really these a_i are the y_i of our normal discussion, and these are $y(x_i)$ and there is some underlying function y(x). Thus, -E translates you to the next term and changes the sign. But then we expand $(1+E)^{-1} = (2+\Delta)^{-1}$ using the binomial series of the last problem, and this leads to a somewhat bizarre looking rearrangement of our series. The terms in the series have the form $\pm (1/2^n)\Delta^n a_0$. This chapter gives us no CLUE as to what good this thing is, but it will return in Chapter 17 where it gets used to "accelerate" the convergence of a summation. In any event, this rearrangement is called **The Euler Transformation**.

In 11.16, again for no seeming reason, we expand the function $x/(e^x - 1)$ and define the coefficients so obtained as the **Bernoulli numbers**. These come back also in Chapter 17 and they appear in various summation formulas!

In 11.17 we ponder the $\mathbf{\Delta^{-1}}$ inverse difference operator. In theory, it inverts one of your little telescoping sum difference equations.

In 11.18 we define $\mathbf{D^{-1}}$ as the **indefinite integral operator** and we can expand Δ^{-1} as a series in D and now we see a use for the Bernoulli numbers right here! If you do the operator algebra, you find that $D/(e^D-1)$ is appearing. Thus, this Δ^{-1} expansion has B_i sitting in it.

Comment: we say how D and Δ were related, the continuous and discrete "derivative" things. So now we also have a relationship between the "integral" things.

In 11.19 we apply the above theory. We consider a telescoping deal $F_k = \Delta^{-1}y_k$ and we use our D^{-1} expansion and we in short order derive the **Euler-McLaurin formula** which is a way of approximating a finite sum by the corresponding integral + correction terms. The integral appears because there is exactly one D^{-1} term in our Δ^{-1} expansion. We will be seeing this one also in Chapter 17.

Now in the above we had $D = \ln(1+\Delta)$ giving us a series expansion for D in terms of Δ^k . In the unworked problems, Scheid goes on do derive some expansions of powers of D in terms of that central difference operator δ. I have not read through these problems to derive these things, but I am sure I could do it. I am not sure whether these things are used later in the book.

****

Chapter 12: Interpolation and Prediction p 79

Problem: Suppose you are given a table of tabulated values of something, and you need to interpolate to a point between two of those values. How would you do it? The answer is that you make a collocating polynomial for a set of nearby tabulated values, and you use that poly to estimate the intermediate value desired. Presumably the more points you fit into your poly, the more accurate it will be at your focal spot of interest. Nowadays we don't have to do this much with computers and calculators, but I see the problem. And I can imagine results of experiments where you might want to do the same thing. This whole subject is then in the realm of "the experimentalist" which is why I probably never learned much about it.

The table shown on page 82 is a great example. This is a table of sqrt(x) values and you want to compute various intermediate values. In problem 12.7 we worry about x = 1.01, and since this is close to the bottom of the table, we use the "forward" Newton formula. This is a no brainer, you just write the first few terms of the formula, and read off the differences from your triangle computed table (table 12.2). You have to use the numbers h and k as outlined above, that is, $x = x_0 + kh$.

In problem 12.8 we use the "backward" Newton formula since we are at the end of the table.

In problem 12.9 we use the Everett formula with its centered δ^2 type differences (see page 41 for all these formulas). In the execution of this problem, we really use Δ^2 instead of δ^2.

As you look at table 12.2, you tend not to believe the Δ^4 data much, since signs change. You are out in the noise here, because the original y_k data is only stated to 5 places. So no point in ever using those distant differences.

Problem 12.10 is an interpolation problem wherein the true δ^2 are stated in the table, so you can use the 2 point Everett formula.

OK, this is a long chapter, but at least I get some of the flavor of it. Practical problems. This is where Scheid is showing WHY you might want to have a collocation formula.

****

Chapter 13: Numeric Differentiation p 97

Once you have decided to fit your data with some collocating polynomial, you can then use that polynomial to estimate your various derivatives. The problem is that small errors in data values can become large errors in derivative values. Fitting values well does not really mean you have fit the

derivatives well. Still, with accurate data you can pursue this way to deduce derivatives, and you can use the same formulas discussed in the previous chapter.

In practice, this chapter does give various series for the derivative y'(x) (and higher ones) where we use our earlier page 40 versions of the Newton collocation polynomial. At the lowest level, you just use a first difference (forward, back, or central). At fancier level, you get Stirling or Newton fore and aft series.

Later in the book, we will use least-squares and min-max to do fits to data that do not go through the data points, but "fit" them in a different sense of minimizing "distance" between y(x) and p(x). Probably the polynomials you get by this method are better for computing derivatives that the collocation polynomial.

In 13.1 we write out the Newton forward formula (a series) and we differentiate it using the trick of the factorial powers, and we then obtain a series for each derivative p(x), p'(x), p"(x) and so on. In these series, we only get powers of Δ, since that is all that is in the Newton forward formula.

In 13.3 we write out Stirling's formula (for the collocation poly, it has δ and μ in it). Again we differentiate, and we get those results. No big deal! We can get all the formulas in this way.

****

Chapter 14: Numeric Integration p 107

This chapter discusses only equal-spacing grid methods. In Chapter 6 p34 we saw the collocation polynomial expressed as Newton's forward formula. Then in Chapter 7 p40 we saw a bunch of equivalent collocation formula with names like Newton backward, Gauss forward & backward, Stirling, Everett and Bessel. These are all rearrangements of the terms, nothing more!

Now, if you want to approximate the integral of y(x) over some range, you replace y(x) with one of these polynomial forms p(x) and integrate a few terms. Some series converge must faster than others!

We start with the Newton forward formula and integrate it for n = 1,2,3 which means we are using the page 34 series with 2,3 or 4 terms. The n = 1 is the linear fit to y(x), the n=2 is a quadratic fit, and so on. The formulas are shown on page 107.

So some methods of numerical integration involve selecting a p(x) collocating poly and then integrating it. You might be able to throw away some of the higher order terms if they are small.

A completely different approach [well, not really different, it is just a piecewise approach, see Newton-Cotes comment below in M&M notes.] to integrating a collocator poly is the **trapezoidal rule**, where you just add up the trap areas and the formula is shown on page 107. But the best all-around method is **Simpson's rule**, where you make a quadratic fit to each (non-overlapping) triplet of points. If you have one triplet, you get the n=2 formula (from Newton) on page 107. If you have 5 points, you have two triplets 0,1,2 and 2,3,4 and you just add the two simpler results and there are then 5 terms in the answer. The number of points must be odd, so the number of intervals is even. This is derived in problem 14.10, and you get the very simple formula for the result. Coefficients are: 141, 14241, 1424241, and so on. Good for hardware! You can always add "correction terms" to your favorite formula. And you have to worry about rounding error as you increase the number of points n+1.

Page 115 illustrates a slight variation. If you have data outside your range of integration, you can do what they say there. Here we are using Stirling with n = 6 which is pretty fancy and accurate. This approach leads to the correction terms for Simpson's Rule.

It is noted that integration is much less sensitive to errors in the data point numbers, then differentiation, as you would expect.

Problem 14.1. Here, we actually integrate all the terms of some of the low-n Newton's forward polynomial fits. We see the first few integrations done. Then Scheid gives us a general table through n=8 more or less. Basically you have an overall constant out front, C, then each of the n+1 y_i values gets a weight called c_i . These are called **Cotes formulas**.

Problem 14.4. For the **trapezoidal rule**, you break interval up into slices, and use the n=1 Newton linear fit formula for each slice. The name is obvious. Notice that if you divide up into 9 slices, say, the formula is not the same as the Cotes with n=8. These are just plain different estimates of the integral.

Problem 14.10. Here we derive **Simpson's rule**. This does an n=2 quadratic fit in each slice, see details already noted above.

Problem 14.17. Here we apply Trap, Simpson, and n=6 Cotes to integrating sinx from 0 to $\pi/2$ which we know is 1.000. We work with a mere 6 data points to model sin(x). Trap says .994 which is PDG. But Simpson says 1.00003 which is astounding. The n=6 Newton is even better at 1.000003. You might expect the more global n=6 thing to be a little better. Also, it happens that the sinx series converges quickly.

Problem 14.15 and 14.16. This is a lead up to the general Romberg idea. Suppose, with some h, you do a Simpson rule and get some integral result A_1 and you know how to estimate the error E_1 . Now divide h in half and do Simpson again, and now you get A_2 and E_2. Comparing the formulas for E_1 and E_2, you see that they have a rough ratio of $(h_1/h_2)^4 = 2^4 = 16$. This is only approximate because $\zeta_1 \neq \zeta_2$, but all we want is an estimate. Once you have computed A_1 and A_2, your estimate is that $E_2 \approx (A_2 - A_1)/15$. So why just add this to your A_2 result to get a better answer. Then you get $B_2 = A_2 + E_2 \approx (16A_2 - A_1)/15$.

In Problem 14.23 this result is made a little more general. Instead of assuming the power n=4 which works for Simpson's rule, just leave the power as n to get the results shown top of page 115. The final action is then to take a programmed approach, and that is **Romberg's Method**. Compute some A_i using trap rule and then correct them as we just said using the n=2 formula. This correction it turns out is the same as if you had done Simpson's rule (which has an n=4 ratio). So the B_i are in effect from Simpson's. You know that to correct them in turn, you should do the n=4 ratio to get some C_i . But then these are as if you had done the Cotes n=4 formula, so next time use the n=6 . In any event, you only had to do a few trap rules to get started, then it is just simple algebra to do all the corrections of the program! Very nice and simple. Notice how fast we get a great result for our sinx integral from problem with a mere 4 point starting position in problem 14.24. This looks very very good to me.

****

Chapter 15: Gaussian Integration p 125

In all previous chapters annotated above, we were worrying about various kinds of polynomials p(x) which could approximate a general function y(x). First we did collocating ones, then we did osculating ones in Chapter 10. We found exact formulas for both.

Let's start in problem 15.1 where we write the osculating poly formula but we assume that we osculate at n points instead of the n+1 points used in the earlier chapter. This p(x) is just written in terms of those U and V functions, and this was Hermite's Formula. If we integrate as shown, we arrive at this fact: [this idea is sometimes called Hermite Interpolation]

$$\int_a^b w(x)y(x)dx \approx \int_a^b w(x)p(x)dx = \sum_{i=1}^{n} (A_i y_i + B_i y'_i)$$

where we see values, slopes, and ugly A_i and B_i objects which at this point are messy integrals.

Now we go at once to problem 15.2 which computes the error in the above estimate of the integral of y(x). Interestingly, it is proportional to $y^{(2n)}(\theta)$. This means that if y(x) is a poly of degree up to 2n-1, there is ZERO error! Remember that our fit p(x) is only of degree n.

This is where I had to sidetrack to understand the application of the **mean value theorem**. You compute the difference of the integrals as shown, where we use the osculating error formula derived earlier. Remember that this error is between two functions, not two integrals. But then we integrate to get the error in the integrals. Remember that ξ is a function of x. In the RHS you treat w(x) times $\pi(x)^2$ as your positive weighting function, and you apply that "generalized integral form MVT" which lets you pull out the function shown and it ends up evaluated at some definite but unknown point θ. We then end up with a computation of our error as shown on top of page 126.

The point is then made: compared to earlier formulas for errors between functions, we now have an error which has the $(2n)^{th}$ derivative, not the nth. This means first of all that there is zero error for any y(x) that is a poly of degree 2n-1 or less! Secondly, in chapters I did not read in Scheid, numeric integration methods with equally spaced arguments only had the nth derivative in their error expressions, so in some sense we are twice as good using the unequally spaced arguments.

At this point we still have our B_i coefficients floating around. Problem 15.3 shows that $B_i = 0$ for any weight function w(x) that has the property shown -- that is, that its integral against $\pi(x)\ x^k$ is zero for powers k = 0 to n-1. In problem 15.4 this is restated as the requirement that $\pi(x)$ be "orthogonal" to all powers or to all polys of an orthonormal set. The idea is that *you want to select* the points x_i such that this condition will be true, since $\pi(x)$ is a function of these points. Problem 15.8 shows a brute force method for computing the points x_i that make this work, and this problem I think shows that such points always exist.

Once you bring in the idea of a set of ortho polys for the w(x) and interval, then you get to apply fancy results relating to those polys. The book works with specific Legendre polys which are correct for w(x) = 1, but I think the results are general to any family of ortho polys. The idea is that if you look at an ortho poly of degree n, call it $f_n(x)$, you can show that it has n zeros, you can just factor it and get that $\pi(x)$ form for $f_n(x)$. This is of course a very specific $\pi(x)$ because you have chosen the special points x_i to get the zeros of $f_n(x)$. But it is always the case that the integral of $f_n(x)\ x^k\ w(x)\ dx = 0$, $k < n$, because the polys are orthogonal and you can always write x^k as a sum of other polys of degree k-1 and lower. **Thus, by picking your special points x_i as the zeros of $f_n(x)$, you are assured that $B_i = 0$**, and you obtain the standard Gauss integration approximation formula which is:

$$\int_a^b w(x)y(x)dx \sim \sum_{i=1}^{n} A_i y_i$$

where the A_i are as shown in problem 15.1. However, this messy result is simplified in problem 15.5 and then further in 15.6, so you end up that A_i is simply the integral of the $L_i(x)$ Lagrange multiplier poly (times w(x)) . Note that this $L_i(x)$ is of course a function of the special points x_i .

Special case of Gauss-Legendre. Here we pick (-1,1) and w(x) = 1 and the $f_n(x)$ are our old friends $P_n(x)$. In this case, the A_i are given by a simple expression shown on page 126 which is derived in problem 15.22. (Scheid is great, everything is derived, I like the approach!) The table on page 135 shows the important numbers for various values of n. For example, for n = 6 you see the 6 zeros of $P_6(x)$, and you see the 6 values of A_i which are pairwise equal.

A simple example is shown in problem 15.24. A trig integral is first converted to the (-1,1) range as shown, and then we use the n=2 formula (two-point Gauss estimate). We know what our zeros are, so we compute the y_i, and we read the A_i from the table and bang, the n=2 estimate of the integral is accurate to an astounding 1/5th of 1 percent! All we had to do is look up the A_i and compute our function at two points! Even the 1-point estimate is good to 10% . It really is in fact amazing.

Now, in our development above, we started with the osculating poly p(x) in problem 15.1, and we ended up with our A and B formula and we then used orthog polys (and zeros of same) to get rid of the B numbers. In problem 15.7 it is shown that you get the same Gauss formula with the same A_i if you just start with the simpler collocation polynomial! So I am now a little confused as to why we did not just do the derivation in this manner. I guess the osculating stuff is needed to derive the formula, then once you have done that derivation, you show that the collocation gives the same result.

General Comments.

(1) The term **"quadrature"** is indeed an archaic form of the word "integration". The idea originally was to take a circle and rearrange it's tiny slices into a rectangle so you could compute the area, a quadrangle. This idea then got extended to other shapes, and finally generally to mean a definite integral. So "Gaussian Quadrature" and "Gaussian Integration" are the same thing.

(2) My Margenau and Murphy book has a small section on numerical integration methods including the Gauss one, and I also have a Dover numerical paperback by Hildebrand. (see notes below) And A&S have a chapter on orthogonal polynomials, but no mention of Gaussian integration in that book. Maybe I really did the general case (arbitrary polys) all on my own in 1978. See also Bateman on ortho theory.

(3) I have a lot of old notes in my math I binder. I derive a formula for A_i in the general case for any known set of ortho polys. I don't think I saw this in any book, but if must be in some book. Of course I don't know for sure that my general formula is correct!

Note Added: In the general case, we have A_i as an integral of $w(x)\, L_i(x)$. Once you have identified your set of ortho polys $f_n(x)$, I think you can use the poly recursion relation along with the Christoffel Identity to derive a formula for A_i in terms of the $f_n(x)$. Scheid does this for the Legendre, but I know that the recursion and Christoffel are generally stated and derived in Bateman. I think I did exactly this in my old typewriter notes, based not on Bateman but on A&S information.

(4) Gaussian integration is an improvement on more elementary numerical integration methods that I should at least know the names of! For example, you might just add up the little rectangles with an equal spacing of n points. You then get

$$\int_a^b y(x)dx \sim \sum_{i=1}^{n} A_i y_i \qquad A_i = \Delta x$$

OK, in notes below (now above!) I have done all this. We have Trap Rule, and Simpson's Rule, and all those things. The error in those formulas is $\sim f^{(n)}(\zeta)$ whereas here it is $f^{(2n)}(\zeta)$, so it is reduced by n differentiations! I think that translates to impressive decimal points.

(5) How do you compare the different ortho polynomial sets? Each set has a different w(x) function, so that will determine which approach you take!

$w(x) = 1$	Legendre	A_i = page 126
$w(x) = e^{-x}$	Laguerre	A_i = page 127
$w(x) = e^{-x*x}$	Hermite	A_i = page 127
$w(x) = 1/\sqrt{1 - x^2}$	Chebyshev	$A_i = \pi/n$, all the same !!!

There is another formula associated with Chebyshev that is completely different and is I think derived in a long Hildebrand section starting page 414. This is another case of w(x) = 1, but is somehow along a different path, I don't think it is in the Gaussian Quadrature world. It's great advantage is that $A_i = 2/n$! So this is a VERY simple formula, though perhaps not very accurate. You still use unequally spaced x values, as shown on page 419 for all allowed cases. [This IS just another Gaussian Quadrature.]

Review: The logic flow of this chapter is not very clear, and in addition, we have the extra, albeit cosmetic-only, confusion that n+1 of earlier chapters is replaced by n here with the point x_0 removed. So here is the logic:

a) assume that p(x) is a poly of degree 2n-1. Expand that poly using Hermite's formula using the U and V functions, and the n y_i and y_i' values and slopes. Use any x_i you like!

b) now integrate both sides of that expansion against an arbitrary function w(x) over some interval. You then conclude that

$$\int_a^b w(x)p(x)dx = \sum_{i=1}^{n} A_i y_i + \sum_{i=1}^{n} B_i y_i'$$

where A and B are given by some messy integrals over w(x) and the $L_i(x)$, all as shown in problem 15.1

c) if you select as your set of n x_i the zeros of the ortho poly $f_n(x)$ [which in turn is defined by the interval and w(x)], then the B_i all vanish. So you now have:

$$\int_a^b w(x)p(x)dx = \sum_{i=1}^{n} A_i y_i$$

// where x_i are the zeros of $f_n(x)$ defined by a,b and w(x)

// and where the A_i can be expressed in terms of $f_{n-1}(x_i)$

Thus, we have replaced an integration of a poly of degree 2n-1 by an n-term sum over products of $y_i = p(x_i)$ and these A weights. The weights have nothing to do with p(x), they are determined by choice of a,b,w(x) and n. The formula is exact as shown. As you increase the complexity of p(x) [ie, increase n], you have to add more terms on the right to keep it exact.

d) Now instead of using p(x), use some general function y(x). Then the claim that the formula is a very good approximation:

$$\int_a^b w(x)y(x)dx \approx \sum_{i=1}^{n} A_i y_i$$

// where x_i are the zeros of $f_n(x)$ defined by a,b and w(x)

// and where the A_i can be expressed in terms of $f_{n-1}(x_i)$

The error in the approximation is proportional to $y^{(2n)}(\theta)$ where θ is some unknown value in (a,b). Reasonable functions have decreasing derivatives, so make n large enough and you get as accurate as you want. This $y^{(2n)}(\theta)$ expression can be used to bound the error for a given y(x) and given n.

e) For comparison, the Trapezoidal Rule has error proportional to $y^{(2)}(\theta)$ while Simpson's Rule gives error proportional to $y^{(4)}(\theta)$. These "rules" can be compared in error to GQ with n = 1 and 2 respectively.

One More Note Added. I really need to write my own notes and do this chapter in the General Case for any w(x). Probably this is all done in Bateman and elsewhere, but I want to "own" my own derivation of the general formulas. I want to drive my own car. Then I can toss those 1978 notes.

Another approach is to leave the B_i terms in there and compute A and B. Then you don't have to use the zeros of the polys as x_i . But, then you need y'_i data at your data points!

The Main Logic of Gaussian Quadrature. (added 11.21.04).

(1) We know how to explicitly construct an osculating polynomial p(x) that passes through an arbitrary set of n points y_i in some interval (a,b), and which matches the y'_i slopes at those same points. This is called the Hermite interpolation, and it has those U and V things. We know that p(x) will be degree 2n-1 or less.

(2) We can brute force integrate p(x) over (a,b) and the result is the A and B summation formula quoted above. This formula is exact regardless of whatever set of n points x_i you pick in (a,b). They could all be over in the left 1% of the interval, for example.

(3) If our p(x) is a pretty good fit to some y(x), then the A/B formula will be a good fit to the integral of y(x) over (a,b). It seems clear that some selections of n points x_i would be better than others if we want to have p(x) be a good fit to the shape of y(x) over the interval. Probably you would put a higher density of your points where y(x) has more activity. If y(x) happens to be a poly of degree 2n-1 or less, then the A/B formula gives the exact answer *no matter where* you position your n points! Notice that as you change your set of points, not only do the y_i and y_i' change, but the A and B change as well, since they are functions of the x_i as well. That is how the sums can still add up to the exact same answer as you change your set of x_i.

(4) There is exactly *one* set of points x_i that you can select which makes all the B coefficients be 0. These are the zeros of $f_n(x)$, which is $P_n(x)$ for w(x) = 1 and (-1,1). If you agree to use these x_i, then you don't get to "select" an optimal set of x_i to match the "activity" of y(x), you have to go with the zeros. So you might imagine this would hurt you if y(x) is highly non-uniform over your interval. On the other hand, you don't have to worry about y'(x) at all, no y_i' values are needed since there is only the A sum to compute. Again, if y(x) happens to be a polynomial of degree 2n-1 or less, this A-only formula gives the exact answer to the integral. For any y(x), the error in the approximation is related to $y^{(2n)}(x)$ [which of course is exactly 0 for 2n-1 polys.]

(5) Consider now the "alternative logic approach" where you instead fit y(x) by a Newton collocation poly (of degree n-1) instead of by the osculating one (of degree 2n-1). Again, you can do this for any n points x_i that you want. To approximate the integral of y(x), you integrate this poly, and you get the A formula with the exact same A's !! This formula is exact for any poly of degree n-1 or less, and you can select the n points anywhere you want in (a,b). If you select your points to be equally spaced, then the A formula reduces to just the Newton-Cotes n-point formula. The hidden benefit of choosing the zeros of $f_n(x)$ is that, if you do so, then the formula magically becomes exact for any poly of degree 2n-1 !! What this generally means is that your approximation to the integral of y(x) suddenly becomes a lot more accurate if you pick those special x_i.

****

Chapter 16: Singular Integrals p 150

Various suggestions are made with how to approach this class of problems. The integral shown as an example has a principle part of 0 (and a complex part of $i\pi/2$ or something like that), so ignoring it gives a completely haywire answer. But sometimes ignoring works.

- ignore
- power series expand, then integrate term by term and look for fast convergence
- subtract the singularity to isolate it into a doable integral, then treat the rest by GQ
- change argument (they claim this is often a very powerful tool)
- add a parameter and differentiate relative to it.

Lots of good examples are given. Typical problem says "do this integral to 6 decimal places accuracy".

****

Chapter 17: Sums and Series p 156

This is a very impressive chapter. I now know that Francis Scheid is a retired BU Prof, but could not find an email address. This book was reissued in 1989 and is on sale today. The more I study in it, the more I like his approach: summarize things in the first few pages, then derive everything in the problems. In this chapter, he uses the topic of "sums and series" as an excuse to describe several pieces of math that I knew nothing about. I could not find any web errata for this book, and found two typo errors on page 165. (One other error is in problem 14.17 page 113).

There are really two topics here. One is how to compute certain finite and infinite sums of series **exactly**, and the other is how to numerically estimate sums of series for which perhaps we don't have exact closed-form sums. In the second topic, we often try to compute something to some specific number "N" decimal places. The Bernoulli stuff is more in the first camp of exact sums (especially of powers), but also for estimation, as in the Euler-McLaurin formula.

I guess I could imagine a similar chapter that discusses exact and estimated *integrals* instead of sums, but I guess that has already been dealt with in the earlier chapters, eg, Simpson's Rule and Gaussian Quad.

Telescoping Series. The statement is made in problem 17.1. The trick is to write your series $\{ y_i \}$ as a difference of something else, like $y_i = \Delta Y_i = Y(i+1) - Y(i)$. If you can do this, then the sum is trivial, it is the next Y value above the upper index limit minus the first Y value, simply because things cancel pair wise. You might think finding $y = \Delta Y$ would be a rare thing, but not so. Every time you see something of this form, you will be able to telescope the sum. On page 22 we have $y = \Delta Y$ for those factorial polynomial things, so you expect something to happen. In problem 17.2 we see an example of adding a finite series of positive powers in this way. The example does 4th power with a fairly complex result that is also in GR p 1. Problem 17.3 shows how to extend the power rule so you can then add any sum of polynomials. Problem 17.4 is just another telescope result, and in this case you can take the upper sum index to ∞ and get a finite result. // I am amazed at this really! I now know how to sum any positive power, and any polynomial, to finite n, and get the exact result. And this is a payoff for that "factorial polynomial" work done back in chapter 4. Three cheers for Scheid.

Rapidly Converging Series. Problem 17.6 gives us the usual sin(x) expansion as an example of fast convergence, so you only need to add a few terms to get a good result. Problem 17.7 does the e^x expansion, but here signs don't alternate so you need a little more work to estimate truncation error.

Alternating Series theorem: I proved for myself this fact: If you have a sign-alternating series where the magnitude of the terms decreases monotonically, then you can say that if you truncate at some point, the magnitude of the error is always less than the magnitude of the first omitted term.

Acceleration Methods. First, in 17.10 we look at a slowly converging series that you need 10,000 terms to get 4 place accuracy. The stunning fact is that you can often rearrange the order of terms in a series to get an another series which, of course, has the same sum, but which converges much faster. In 17.11 we use the **"Euler-McLauren transformation"** [which we derived on page 75!] to do our rearrangement of the 17.10 series. This applies only to a sign-alternating series situation. The result is a series in the Δy objects which does in fact converge rapidly. In this example, only 15 terms of the rearranged series are needed to get 4 places, compared with 10,000 terms of the original series. This is really amazing! Problem 17.13 just shows that a certain log power series expansion gets slow in a certain range of x. But I think problem 17.14 then shows a way to "accelerate" such a slow series.

Subtraction (aka The Comparison Method). Problem 17.16 gives an example. We have an example of a slowly converging infinite series (needs 5745 terms for 3 place accuracy!). We write it as a difference of two series where the first one is a standard one that we know how to sum, and the residual series converges fast, need only 10 terms now. This is a very powerful idea and I will keep it in mind.

Bernoulli Polynomials and Numbers. Suddenly in problem 17.20 we are off on what at first seems a distraction. We start off with the definition of the B-polys $B_i(x)$, and the notation used here is the same as

GR and AS. Scheid then goes on to think about $B_i = B_i(x=0)$. These B_i are called "the Bernoulli numbers" by GR on page 1078 and by AS page 804. However, Scheid reserves the term "Bernoulli numbers" for $b_i = (-1)^{i+1} B_{2i}$. He does this for two reasons. First, the odd B_i are all 0 starting with B_3. Second, the even B_i alternate in sign. He removes this alternation with his factor, so all his b_i are positive. I admit the b_i are nice, but it is hard to fight with the big guns.

So why do we care about B polys and numbers? He first shows that the numbers B_i have some very strange properties that require a special notation to understand. here is the first one:

$(B+1)^k = B^k$ // where use binomial expansion and then replace B^k with B_k both sides.

What this really says is that all the non-leading terms on the LHS add up to 0. You can use this to iteratively compute the B_k for k = 0,1,2,3... and he does so.

Next, in 17.21 he shows that you can write $B_k(x)$ as a power series in x, and then the B_i are involved in the coefficients as shown. This lets us write down the first few polys as shown. All standard notation.

Next, in 17.22 we have a very major result: $B_k'(x) = k\, B_{k-1}(x)$. In AS, only the extended Hermites seem to have this same property. It lets you do integration by parts to develop interesting results as we shall see.

Next, in 17.23 we see that $\Delta B_i(x) = i\, x^{i-1}$ where $\Delta B_i(x)$ means $B_i(x+1) - B_i(x)$. We can see right away that this is a powerful result, because now powers of x can be summed by telescope! [But we already had a way to do that, albeit harder, using the factorial poly expansion, see Telescope above.] Note that if we set x = 0, we find that $B_i(1) - B_i(0) = 0$ so $B_i(1) = B_i(0)$.

Next, in 17.25 we learn that on (0,1), all the $B_i(x)$ integrate to 0.

Next, in 17.27 we learn that $B_3 = B_5 = B_7 ... = 0$, above 3 all odds vanish.

So, problem 17.29 shows how to do a **finite sum of x^p** -- a simple telescope. Now finally I see why the Bernoulli numbers show up in such sums, I have at least seen them before! [So how we know the sum of an arbitrary x^p to n in trivial closed form, much better than our earlier method in Telescope, but really the exact same idea with a new layer of notation.]

Problem 17.30 is a bit of a digression. We see just quoted some expansions for $B_n(x)$ which "extend" this thing to be periodic over all x, so things in the (0,1) range just repeat, sort of like aliasing Using this expansion, we get a **formula for a finite sum of $1/x^p$ where p is even,** and the Bernoulli numbers appear in these sums as well. There does not seem to exist a similar formula for odd p. [Scheid has a very rare typo in these two formulas, I have penciled in the n! missing factors.]

In 17.32 he derives an asymptotic expression for b_i with large i.

So to conclude: The Bernoulli numbers and polys allow you to compute finite (and infinite if they converge) power series sums in compact closed form!!

The Euler-MacLaurin Formula. This is not to be confused with the E-M "transformation" mentioned earlier (derived page 75). Here you do parts integration using the nice fact that $B_k'(x) = k\, B_{k-1}(x)$, and you end up with a way to write a sum as an integral plus some extra terms. [In fact we did this once before on page 76 using an operator expansion, but without error estimate.] On page 167, the first underlined form is used in problems 34 and 35 to compute things. Notice that k is any integer you want in the range k = 2,3,4... (so this is indeed a strange formula). The larger you make k, the smaller the error in general, unless the error happens to be zero at some point, then it stays zero. The error E_k is also shown here.

This is sort of a "subtraction" approach like that discussed above, but the "known sum" is in this case an integral. Problem 17.34 shows how you apply the formula with k = 2. Problem 17.35 uses the formula

again to compute an infinite sum limit which defines C = Euler's constant. Problem 17.36 is the truck and fuel across the desert problem and again the E-M formula is used to estimate an answer.

Wallis Product. We derive in 17.37 a strange product formula for $\pi/2$ and for sqrt of same. You think of each formula as a limit. This formula finds use in the Stirling problem 17.39, otherwise an oddity.

Stirling's Series for large factorials. The result is given on page 170 top. For large n, you can set the RHS = 0 (which makes the inside of the log be 1) and get the usual formula for n! for large n, but the series is more accurate for smaller n. This series is our first example in this chapter of an "asymptotic series", which is in fact a non-convergent series.

Asymptotic Series. I am not sure I remember this from my math courses. The definition is shown underlined in red. Consider $S_n = \Sigma^n a_i (1/x^i)$, and we are going to be interested in two limits: $n\to\infty$ and $x\to\infty$. First, for some value(s) of x, it may be that the series converges to some f(x) as $n\to\infty$. If it does so on some domain of x, we call that uniform convergence over that domain.

Suppose, however, that f(x) *diverges* for a region of x that includes $x = \infty$. Consider the following quantity: $| f(x) - S_n(x) |$. For a value of x in our bad region, what can we say about this quantity: $\lim_{n\to\infty} | f(x) - S_n(x) |$? Think of f(x) as something like e^x and $S_n(x)$ is some series that diverges. We would have to say then that $\lim_{n\to\infty} | f(x) - S_n(x) | = \infty$.

Fine. Now suppose it happens that $| f(x) - S_n(x) | \sim K_n / x^{n+1}$ for large x and for a specific n. Then for that value of n, you can drive the error under any ε by making x large enough. You need $x^{n+1} > (K_n/\varepsilon)$.

Now suppose the above is true in fact for all positive integers n. For each value of n, you can make the above statement: for any ε, you can get under the error for a sufficiently large x : $x^{n+1} > (K_n/\varepsilon)$. However, for some *fixed* large x, you cannot drive the error under some ε by making n large, because if that were the case, the series would be convergent, and we have assumed it diverges. It must then be that eventually, as n increases, the K_n start getting larger and, even though x is large, x is fixed and the error starts to increase. That is, $\lim_{n\to\infty} [K_n x^{n+1}] \to \infty$ for any large but fixed x (in our region of series divergence). If you are lucky, for your specific large x of interest, the error will go down for the first few terms, but then eventually it must go up again. In this case, for a given x, there is an optimal n, and there is nothing you can do to get the error to be less than that minimum amount.

If $| f(x) - S_n(x) | \sim K_n / x^{n+1}$ for large x, then $x^n | f(x) - S_n(x) | \sim K_n/x$ and therefore $\lim_{x\to\infty} x^n | f(x) - S_n(x) | = 0$. If this is true for all n = 1,2,3..., the $S_n(x)$ is called an **asymptotic series**.

I am unsettle just a little here, and I think there is more depth to this subject that is presented by Scheid or other sources I could find easily.

(1) Would it be possible to have $| f(x) - S_n(x) | \sim K_n / x^{n-1}$, or perhaps $\sim K_n / x^{n+2}$?

(2) The K_n / x^{n+1} is compatible with the notion that the error in your series is like the first omitted term.

(3) Is it always true that you can get K_n by assuming that the error is less than the next term, say for alternating series? We know this is true for a convergent series, and we have seen some examples of asymptotically convergent series where it is true. Maybe you could consider some region of convergence in x and then sort of continue the conclusion (error less than first omitted term) out of that region to a region where the series only asymptotically convergent.

(4) Can you prove that there is some optimal n for a given x? This is proved for specific examples, but I don't see a general case proof. It is just a claim.

I leave these as questions for now. Stirling is an example of an asymptotic series.

****

Chapter 18: Difference Equations

What are these, how do they arise, and how do you solve them?

What are these?

You might imagine using Δ and Δ^2 and so on as analogs to D and write your difference equation as shown, but this is not done. The reason is that in the end, it all boils down to a sum of $y_k = 0$, so you usually put the term with the largest index on the LHS and the rest on the RHS.

The **order** of a difference equation is the max range of the y_k that appear, not the power of Δ that might appear in the Δ form, though these would tend to be related somewhat.

An example of a difference equation with constant coefficients is $y_{k+2} = y_{k+1} + y_k$ -- it is second order. The solution for a certain y_0 is the set of Fibonacci numbers, it turns out.

How do they arise?

This kind of question is kept beyond the scope of Scheid's book. Obvious situations are those with a **discrete time step**, such as biological generations, things that happen in a lumpy way. Any problem with a well-defined discrete step could lead to a difference equation, but this is just not discussed here. In contrast, a differential equation arises in the continuous world of physics, or a world that is approximated as being continuous.

However: difference equations **also arise** when you approximate a differential equation for numerical computation purposes, and this is the subject of the next chapter.

How do you solve them?

Even if the equation is non-linear, like $y_{k+1} = 3\, y_k + y_k^2$, the mechanical iterative method of solution always works. **You can write a perfect program to solve ANY difference equation exactly**. You assume some initial value for y_0 and then off you go: y_1, then y_2, and so on. So there is never really a problem in solving the equation. BUT, you don't get any analytic insight as to the large scale behavior! And you cannot see how the solution behaves as you vary a parameter, for example.

For linear equations of first and second order, you can solve by standard techniques. To wit:

In the world of *differential* equations, you can restrict to linear, homogeneous, and constant coefficients (but any order). As shown in M&M page 49, there is a closed-form answer for the solution. In this continuous-x world, the solutions are $y(x) = \exp(r_n x)$ where r_n are the roots of the **characteristic equation** (which you just factor to get those **roots**). If two roots coincide, you get $\exp(r_n x) * x$ as the second independent solution for that pair of identical roots, and similarly for larger sets of coinciding roots.

In the world of *difference* equations, the characteristic equation is the same, the roots are the same, and the solutions are $y_k = (r_i)^k$ where r_i are the roots, and when two roots coincide, $y_k = k\,(r_i)^k$ is the second solution. All the ideas of linear combinations and stuff carry over.

You can always think of an x coordinate as $x_k = x_0 + hk$ where k = 0,2,3,4 and h = spacing. We always think of an even grid spacing on x in this world.

An example of a linear second-order difference equation with constant coefficients is $y_{k+2} = y_{k+1} + y_k$ -- it is second order. The solution for a certain y_0 is the set of Fibonacci numbers. Each number is just the sum of the previous two numbers.

Partial Tour of the Problems.

First Order Difference Equations

18.1 Solve $y_{k+1} = ky_k + k^2$. The first comment I make here is that this has "non-constant coefficients" but is linear and first order. You could rewrite it in terms of $x_k = x_0 + hk$ if you wanted. The solution is that you just iterate and you are done.

18.2 Solve $y_{k+1} = a_k\, y_k + b_k$. This equation is first-order, linear, non-constant coefficients, and non-homogeneous: it is driven by the function b_k (rewrite as $y_{k+1} - a_k\, y_k = b_k$). A solution is found as the series shown. If $a_k = 1/z$, then $p_k = (1/z)^n$ and then the solution y_n would be a polynomial in z. So you can see how the solution of a difference equation can end up being a polynomial in some "parameter" z. More on this later.

18.3. Special case of homogeneous ($b_k = 0$) and constant coefficients ($a_k = r$). Here we get the expected power answer $y_k = A\, r^k$ -- here r is the only root of the characteristic equation since first order.

18.4 Same as above, but driven by $b_k = 1$. We get a closed form solution.

18.5. Same as above but b_k is written as c_{k+1} for the driving term, and replace r with x. If you solve this little difference equation in the usual iterative manner, you find that you are simply "evaluating" a polynomial in x whose coefficients are the c_i in an *efficient manner* that minimizes the number of multiplications required. If you evaluate naively, you need n multiplies to build up the powers of x, and then another n to mult in those coeffs. In the efficient method you only need n multiplications total, and this is **Horner's Rule**.

18.6. Here we insert parameter "1/x" in one of our constant coefficients a_k (as done above) and we drive the with 1. We get as before a poly in x as the solution which for large k approaches exp(x). [Here, x is just a parameter we called z above, it is not $x = x_0 + hk$.] I guess it is interesting that you can write the series expansions of a function like e^x as the solutions of a difference equation in which parameter x appears. Not sure how I would make use of this fact.

18.7 Similar to 18.6, but now your solution poly approaches sinc(x).

Digamma Function

18.8 Recall that we could sum up series (finite or otherwise) using the **telescoping method** if we could write the summand as a difference, $y_{k+1} - y_k = b_k$, where you were trying to sum the b_k. **But** this is a first order linear difference equation driven by b_k . In the special case that $b_k = 1/(k+1)$, Scheid shows that the solution y_k is exactly $\psi(k)$, the Digamma function, which you write as a certain infinite series. Then of course a sum [0 to n-1] of $1/(k+1)$ can be done telescopically and the result is $\psi(n) + C$, so this series can then be used as a statement of $\psi(n)$. But using the first definition works for complex x in $\psi(x)$. You think of $\psi(x)$ continuing $\psi(n)$ as you think of $\Gamma(x)$ continuing (x-1)! [This is similar to what we did with those Bernoulli polynomials back in Chapter 17 and the sum there was of k^p . There we got a difference of Bernoulli numbers, while here we get a difference of digamma functions.]

Back on page 178 Scheid shows why in a sense $\psi(x)$ is analogous to log(x+1).

18.9 In the previous problem it was shown that **$\psi(x+1) - \psi(x) = 1/(x+1)$**, that difference idea. This lets you trivially compute the sum [1 to n] of $1/(k+a) = \psi(n+a) - \psi(a)$.

18.10. Here he derives the famous sum that makes $\psi(x)$ analogous to $\log(x+1)$. The starting point is always to write as partial fractions and apply previous results.

18.11 By computing $\psi'(x)$ and knowing $\psi'(0) = \pi^2/6$ from earlier work, he writes a simple formula for **$\psi'(n)$**. So this makes use of an earlier Bernoulli sum we did.

18.12 Another messy-looking series is evaluated (a sum of rational polys) by partial fractions in a very clever way to give digammas of integral arguments which in turn add up to a simple numerical result $\pi^2/6$.

18.13. A very strange sum is added up, each term is the sum of squares of integers to that point.

18.14. Here we see that **$\psi(x) = \Gamma'(x)/\Gamma(x)$**. See page 258 of A&S, also called the Psi function.

The next set of problems in this chapter looks at the linear, second-order, homogeneous case with constant coefficients, including some Fibonacci number problems. Then we have some non-homo problems, then some boundary value problems, then some non-linear examples, and finally a few problems on higher than second order. I have really only commented on problems I have done! I am not heavily interested in difference equations right now, so I skip this stuff.

****

Chapter 19: Differential Equations

For linear differential equations lots of theory exists. In this chapter, we consider ONLY the following differential equation : $y'(x) = f(x,y)$. This is first order and in general NON-linear. In fact, this is the most general first-order differential equation that you can write! The most general second order would be $y''(x) = F(x,y,y')$.

The first-order ODE (Ordinary means one variable, x) $y'(x) = f(x,y)$ is called "the classical problem". As in the last chapter, Scheid does not talk about where such equations arise, he just wants to solve them. In M&M page 33 we are given a set of 9 examples of first-order ODE's arising in physics, and most are in fact very non-linear.

So, how do you solve $y'(x) = f(x,y)$ in some general manner?

The **method of isoclines** is an interesting way to visualize the general shape of a solution $y(x)$ graphically. You set $f(x,y) = M$ for various M values and plot these as dashed curves. Then starting at some chosen point, you draw $y(x)$ such that, when it crosses a dashed curve, at the point of crossing the slope is equal to the M value of that dashed curve. Draw this as a solid curve. Start at some other point, and you get some other solid curve. These are the solutions! Some solutions can be very strange as noted.

The **Euler method** is this: approximate the ODE by setting $x = x_k$ on a fine mesh, and then $hy'(x_k) \approx y_{k+1} - y_k$. So you now have $y_{k+1} = y_k + h\, y'_k = y_k + h\, f(x_k, y_k)$. This is a first-order difference equation that is in general non-linear. We said earlier we can solve ANY difference equation iteratively, and this one is no exception. So you start with some initial value and iterate and get your numbers y_k. The hope is that if

you set h small enough, then your answer will agree within ε to the exact answer of the exact ODE. This is done in detail in problem 19.3 and 19.4. In practice, the method is not very efficient because you need very small h if you are going to "go" very far from the starting point. [When solve an ODE analytically, you don't normally think in terms of *moving along in x* (starting at some x_0) to map out the solution y(x). But that is really what a differential equation is all about, and this is made clearer in this difference equation approximation.]

Problem 19.5 - 19.8. Here we use the Euler method to prove a theoretical result, namely, that the general first-order ODE y'(x) = f(x,y) has a unique solution for each starting point y(0). Certain reasonableness assumptions are needed for f(x,y), such as the sort of analytic "Lipschitz condition".

The **Taylor Method** is more practically useful. You know y'(x) = f(x,y), so you can compute as many higher derivatives as you want, perhaps going through 4th. You then use these to expand y(x) around some point x, say, so you can say $y(x+h) = y(x) + h\, y'(x) + 1/2\, h^2\, y''(x)$ + etc. If you keep only the first term, then you are back to the Euler method. Say this all again

$$y(x+h) = y(x) + h\, f(x,y) + 1/2\, h^2\, f'(x,y) + \ldots$$

Of course in computing something like f '(x,y) you need to do the usual chain rule and things can get pretty messy (unless you have Maple handy, perhaps). That is,

$$f'(x,y) = d/dx\, f(x,y) = \partial f/\partial x + \partial f/\partial y\, y'(x) = \partial f/\partial x + \partial f/\partial y\, f(x,y)$$

So you end up on the Taylor right side with somewhat of a mess. In the example shown in problem 19.10, all four derivatives are computed for the example $f(x,y) = xy^{1/3}$.

So, basically this is an improvement on the Euler method. Instead of assuming a straight line fit at each point, here we assume a polynomial fit of degree 4 say. You do things step at a time by repeatedly using the same formula, but you replace x with x+h in on the RHS in the second step (and similarly for y), and so on. In your computer program, you would code each derivative as a function of x, and then just feed in the up-stepping values as you go, and compute a new y(x+h) in each step. The table shown on page 201 shows how the error builds up relatively slowly as you move a fairly large distance. The key idea is that you are doing a good polynomial fit on an as-you-go basis. [I suspect that almost everything in numerical analysis boils down to polynomial fits.]

Problem 19.11 does this same Taylor idea in a different way. He assumes a power series in x solution with unknown coefficients, jams it in, and comes up with recursor relations for the coefficients, just as we always do with the classical physics ODE's. In this example, the coefficients happen to come out very simply and a closed form for the series can be written which is in the limit the exact solution.

So what is **Runge-Kutta** (1895 and 1900) ? It is a variation of the Taylor idea. Instead of having to compute the derivatives of f(x,y) to 4th order, you get the same result by evaluating f(x,y) at a mesh of neighboring points of the form f(x+δx, y+δy), but in an iterative manner as shown in the equations on page 202. There is a different RK method for each different set of values you start with {m,n,p} and these in turn I think imply the coefficients { a,b,c,d }. In effect, you are really computing the same Taylor expansion information, but you do it with numerical derivatives instead of continuous ones. This means your program only needs to know how to evaluate f(x,y), it never has to compute derivatives. I can then imagine a sort of general purpose RK program where you supply just the evaluation function f(x,y).

The "classical" values of the constants are shown top of page 203. Now for each step, you evaluate your f(x,y) four times, and then you do four additions and you then have a very similar y(x+h) that you would have gotten from the Taylor approach.

Maple knows about R-K in some particular implementation (1976, in a book by 3 people), look up Runge in Maple help! It can do partial differential equation solutions as well as ordinary using RK.

Problem 19.14 solves our prototype chapter problem using R-K, and it is better than Taylor.

Scheid derives the RK method on page 202, but I did not go through the messy details.

On page 204 I think he wants to show that these Taylor methods actually "converge" in the sense that you can get any desired accuracy by making h smaller.

Finally, we have a class of "**predictor-corrector**" methods for solving our "classical problem". Page 194 shows a very simple example: use Euler as your predictor first shot, then improve that with the corrector expression shown. The main idea is that you use the predicted y_{k+1} in the corrector's expression for y'_{k+1} which is just $y'_{k+1} = f(x_{k+1}, y_{k+1})$. I think this is a bit like using Simpson's rule on individual segments. Here we do a little iteration at each value of x, then we move on to the next. In the first example, the predictor is a linear fit, then the corrector result appears quadratic. This is really shown in problem 19.22.

The predictor shown in problem 19.23 spans four mesh points and looks Simpson's rule like to me.

The **Milne method** uses the predictor and corrector shown on page 209. Then we have the **Adams predictor** on page 210. A general form predictor is shown bottom of page 210. Basically, as you add more terms, you get a better first shot before you apply your corrector, the general form of which is shown on page 211. The rule in a predictor formula is that you have y_{k+1} on the left, and everything on the right has to be index k or lower. The corrector however is of the same form but is allowed to have y'_{k+1}because you can compute this as $y'_{k+1} = f(x_{k+1}, y_{k+1})$ and you use here the predictor's y_{k+1} value.

Convergence and Error analysis occupy much of the pages in this chapter, and I have ignored all this stuff in my pass here, but I agree that these are essential topics.

****

Chapter 20: Differential Problems of Higher Order

Two major claims are made and proven, I am a little surprised to learn these simple facts:

- You can solve any *system* of first-order equations by methods of Chapter 19. (intro)
- Any higher order ODE can be rewritten as a system of first-order equations! (problem 20.5)

Thus, in fact, all higher order problems have well-defined numerical methods. [But I guess we knew that from the claim made in Chapter 18 above which said we can solve any difference equation exactly.] In addition to these claims, certain other methods are discussed in this chapter:

- Infinite **Series Methods**, with and without **perturbation** in a small parameter
- Numerov's method for special-case equations

Problems showing how to solve systems of first order ODE's:
20.1 shows how to solve a 2x2 system of equations by doing Taylor series expansions of the functions.
20.2 shows how to do a 2x2 using a 2x2 extension of the Runge-Kutte method
20.3 shows how to do a 2x2 using the Adam's predictor/corrector method.

Problems showing how to reduce higher order equations to systems.
20.5 The general case is done here! Each derivative is regarded as a separate function.
20.6 Treating a special case where first derivatives don't appear. End up with a 6x6 set.
20.7 Apply the 2x2 Runge-Kutta method to a second order ODE "van der Pol".

Problems solved with series expansions (here, we are not using any difference equation stuff)
20.8 Just an example of doing series and matching terms, linear second order
20.9 Same idea, but with a non-linear second order example
20.10 Solve **Bessel's** equation by series, get formula for $J_n(x)$
20.11 Show $J_n(x)$ series convergence using **The Ratio Test**.

Remaining problems deal with getting an asymptotic series for Bessel, doing perturbation in the right way, and finally doing an application of **Numerov's Formula**.

****

Chapter 21: Least Squares Polynomial Approximation

We have now concluded the set of Scheid chapters (18,19,20) which treat difference and differential equations. That was one large topic. We are now starting a new section of the book which in a sense continues earlier chapters on fitting y(x) with a polynomial. In those chapters, we did poly fits by finding a poly p(x) which passes through a set of n or n+1 points (and in the Hermite case we matched derivatives as well). In this and the next chapter, we find p(x) fits that miss the points, but are in some sense better fits nevertheless.

This chapter has a HUGE number of worked and unsolved problems! The real MEAT of this chapter is (as usual) in these "worked problems", so I will plow ahead through them now:

DISCRETE DATA, THE LEAST SQUARES LINE

21.1 Here we consider a "least squares" linear fit $p(x) = Mx + B$ to N+1 data points $\{x_i, y_i\}$. This example shows how the funny sums appear from doing derivatives with respect to the parameters. For example

$$s_0 = \Sigma (x_i)^0 = N+1 \qquad s_1 = \Sigma (x_i)^1 \qquad s_2 = \Sigma (x_i)^2$$

Here, each of these sums is a sum of certain powers of the x-data only. You need another set of sums

$$t_0 = \Sigma y_i (x_i)^0 = \Sigma y_i \qquad t_1 = \Sigma y_i (x_i)^1 \qquad t_2 = \Sigma y_i (x_i)^2$$

These things naturally appear in your D=0 equations. In our little m=1 case here (linear fit) we have m+1 = 2 parameters, and so we have 2 equations to solve, and we do so and we find M and B as on top of page

240. One could imagine doing this for m = 2,3,4... etc to get higher order fits, where m is the degree of your polynomial.

21.2 Top of page 241 shows some scattered data relating to **golf**, and I have drawn in the x values which label the 10 data points. We apply the solution of problem 21.1 to the data and find the linear fit shown!

Our D = 0 equations are *coming from* the fact that we are requiring that our "least squares difference sum" be minimized by our poly fit. This sum is S on page 241, where m is the degree of the polynomial. The general problem has coefficients a_i , but our special case uses M and B above for m = 1. I don't think I have ever thought about least squares ortho poly theory with discrete data.

21.3. Once we have the linear fit to the experimental data, we can replace that experimental data with better data that all lies on the curve. We just map each point vertically to the curve (line in this case).

21.5 Here we have a table of experimental data that we wish to fit with a rising expo (rather than with a polynomial), and we have A and M as parameters to be fit. Using log, we turn this into linear and make a new table, then we use linear formulas to get a fit. Issue is what you are really minimizing here.

DISCRETE DATA, THE LEAST SQUARES POLYNOMIAL (theory, then apply theory)

21.6 Here we do the general case of polynomial fit, and D = 0 produces a set of m+1 equations as shown in terms of the partial sums s_k and t_k. These are the **normal equations** (this phrase comes from the general theory see below) It is noted that this set of equations is "poorly conditioned" so we will not solve them by Cramer's or such for large N, stand by.

21.7 Our first Stakgold type **theory discussion**. Here Scheid assumes that we have a Hilbert space (apart from completeness), because we have a norm obtained from an inner product, and a metric obtained from that norm. (I just finished reading this stuff in S yesterday). He shows that there is a unique projection of an arbitrary vector **y** in E onto a subspace S, and the projection vector here is **p**. Any other vector **q** in S is farther away from **y** than is p. He proves this fact by working with an assumed orthogonal basis $\{e_k\}$ in S.

21.8 Now we have a *non*-orthogonal basis $\{ u_k\}$ for S. In this case, vector p is still there (the projection) and you want to find the coefficients a_k of p. Due to the non-orthog, you end up with a set of m+1 equations that you can solve for the a_i. The corollary is simply this: if you have a larger basis for all of E that includes $\{ u_k\}$ for S, you can do your projection onto p from y by simply truncating the terms as shown. By the way, the "system of equations" looks similar to that we got in the LS solution.

21.9 **Now we apply our general theory**. Let E = space of function *values* $f(x_i)$ for i = 0 to N. So E is not really a space of continuous functions here, it is more like an E^n space, where a vector is in this case N+1 numbers. We interpret these "coordinates of the vector f" as function values at x_i. So perhaps $f_i = f(x_i)$ would be a good notation for the vector **f**. Our point **y = f** in E has coordinates y_i , and these numbers are the experimental data of our least squares fit to come.

Next, S = space of function values $p(x_i)$ where the function p is a poly of degree m or less. Obviously this is a sub-space of E. What about a non-orthogonal basis for S? We can use $[\mathbf{u_k}]_i = (x_i)^k$ for k = 0 to m. Here, k is labeling the basis functions, while i is the vector space coordinate index. Basically, our basis functions are the usual continuous "powers" , but evaluated just at the N+1 points x_i .

How do we then write our polynomial fit? $\mathbf{p} = \Sigma\, a_k\, \mathbf{u_k}$ sum from k=0 to m. Note use of bold to show vectors. The components of vector **p** are $p_i = \Sigma\, a_k\, (x_i)^k$

Now comes the big payoff.

First, the least squares sum thing is seen to be simply $\| \mathbf{y} - \mathbf{p} \|^2$, so we are going to minimize the distance between **y** and **p**, and that of course is done by making **p** be the projection onto space S which has that powers basis. Then **p** will be the best fit to **y** given that you have to stay in the subspace S.

Second -- and here is the main act -- we know that $\langle p,u_k\rangle = \langle y,u_k\rangle$ because $\langle y-p,u_k\rangle = 0$ since y-p is perpendicular to S. We write out $\langle \mathbf{y},\mathbf{u_k}\rangle = \Sigma\, y_i\, (x_i)^k$, and then we write $\langle \mathbf{p},\mathbf{u_k}\rangle = \langle \Sigma\, a_j\mathbf{u_j},\mathbf{u_k}\rangle = \Sigma\, a_j \langle \mathbf{u_j},\mathbf{u_k}\rangle$. But of course this last inner product is $Q_{jk} = \Sigma_s\, [\mathbf{u_j}]_s\, [\mathbf{u_k}]_s = \Sigma_s (x_s)^j (x_s)^k = \Sigma_s (x_s)^{j+k}$. We have then shown that:

$$\langle \mathbf{p},\mathbf{u_k}\rangle = \langle \Sigma\, a_j\mathbf{u_j},\mathbf{u_k}\rangle = \Sigma\, a_j \langle \mathbf{u_j},\mathbf{u_k}\rangle = \Sigma_j\ a_j\, \Sigma_s (x_s)^{j+k} = \langle \mathbf{y},\mathbf{u_k}\rangle = \Sigma_i\, y_i\, (x_i)^k$$

Notice inner products sums go 0 to N (because the inner product is defined in E), but a_j sum goes 0 to m since a_j are coefficients of p in S. Now define

$s_{j+k} = \Sigma_s (x_s)^{j+k}$ or $s_r = \Sigma_s (x_s)^r$ and $t_k = \Sigma_i\, y_i\, (x_i)^k$

Then we have this set of equations,

$\Sigma_{\mathbf{j}}\ a_{\mathbf{j}}\ s_{\mathbf{j+k}} = t_{\mathbf{k}}$ where sum is j = 0 to N and k = 0 to m.

So we have m+1 equations. Let's write some of them out.

k=0 $a_0\, s_0 + a_1\, s_1 + \ldots + a_m\, s_m = t_0$ // note that $a_{m+1} = 0$ and so on up to N.

k = 1 $a_0\, s_1 + a_1\, s_2 + \ldots + a_m\, s_{m+1} = t_1$

....

k = m $a_0\, s_m + a_1\, s_{m+1} + \ldots + a_m\, s_{2m} = t_m$

These are exactly the "normal equations" we found in problem 21.6 by setting D = 0. So we have now derived the least squares solution equations in the "correct" manner. The equations are really just coming from the idea of a projection and a non-orthogonal basis. Excellent!

Now before going on, recall from Stakgold how you have "rearrangement problems" with "changing coefficients" when you use a non-orthogonal basis. I'll bet Scheid is going to get us to an orthogonal basis very soon here.

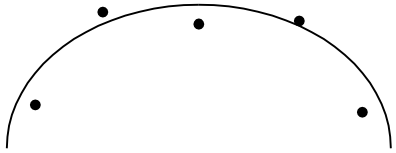
21.10. Now we take the same golf data from 21.2 and this time we do a quadratic fit instead of a linear one. We compute the new s_3 and s_4 and t_2 . Note that as you "add one more to m", you add one more t, but you add two more s, since we have that s_{2m} sitting in the last equation above! He carries it through and you see that the quadratic correction term is quite small, since the linear fit is quite good as the picture shows.

SMOOTHING AND DIFFERENTIATION

21.12. Suppose we have 5 data points and we want to do a quadratic fit. Our fitting poly is as shown in terms of t, where t = 0 for the central point of the five points. The a_i are then shown. Notice the

combination $\delta^4 y_k$ which appears in a_0. This is a difference like that shown on page 15, but it is "central" not "going to the left". The δ stuff appears on page 40 then page 44 shows that $\delta^4 y_k = \Delta^4 y_{k+2}$, so yes, it is just the shifted 4th difference.

21.13 Here we think again about the previous problem. Here is a parabola fit to 5 data points:



Based on this, as with the golf example, you really should vertically translate the dots (experimental data with error) so they fit on the parabola. Since the parabola in t has $p(0) = a_0$, we use the result of the last problem to say that the central point should really be at $y_0^{\text{corrected}} = y_0 - (3/35)\, \delta^4 y_k$.

21.14 Here we have 11 data points with random error added. They should fit to y = sqrt(x) for x = 1,2...10. We draw a picture showing the various δ type differences. We do enough so we get down to the δ^4 ones as shown, then we compute the 5-point central correction as in previous problem, and we compute the corrected values in the bottom row. So this is just an exercise in "smoothing data".

21.15,16 This is a detail about how to handle the two end-values in a table where the normal method above does not have enough data. I skip it.

21.17 Here author shows that the smoothed data has about half the RMS error that the original data in the example we have been working with.

21.18. We know that $p'(0) = a_1$ so we can use this as an estimate of the **derivative** of the parabola at the central points. By using the parabola, we are of course using the "corrected points" to do our derivative estimate. The h is added just to allow for non-unit x_i spacing.

21.19 In this problem, we use this idea of smoothing the data first, *then* doing the derivative estimate, which is what the previous problem did. We compare this method against an earlier formula we had where no data smoothing was done. We do this with that noisy square root data shown in problem 21.14. Since y = sqrt(x) is the exact thing, we know the exact y'(x) as in the last row. The smoothed approach is in fact somewhat better.

The rest of this section is more of the same with extra twists, I shall skip these problems and move on.

ORTHOGONAL POLYNOMIALS, DISCRETE CASE

21.24 A fascinating fact. Back on page 241 we see our simultaneous equations, and we know we can write them in matrix form SA = T and we want to invert this matrix S. However, when N is large, this problem trivially shows that the matrix S is approximately equal to the **Hilbert Matrix**, something I never heard of before today! Author claims it is "ill conditioned" and will discuss this later in the book. But a quick web scan shows that the Hilbert matrix has a very bad "matrix condition number" on the

order of 10^5 for as small as a 5x5 (well conditioned has number = 1). This number measures how the determinant varies with small errors! So this problem is just showing that if we try to do a straight Cramer's Rule solution of our least squares fit problem to compute the a_i , we are going to have numerical problems!

21.25 The solution to our ill-conditioning problem is to replace the non-orthog basis with an orthog one (finally we are getting to this!).

21.26. Now we face the problem of constructing the orthonormal basis functions. These are going to be Legendre-like animals, BUT we are in the discrete world where we want functions evaluated at our x_i points which are **equally spaced**. Orthogonality does not involve the usual integral from -1 to 1, it involves the sum over the x_i as in our "general theory" on page 243. Here we replace the x_i with unit spaced values t = 0,1,2....N, so we have N+1 points of interest.

The ortho process is not well presented in my opinion. Instead of dealing with powers t^n , as we often do in the discrete world we use those "factorial polynomials" $t^{(m)}$ in the derivation. I could grind through the derivation and obtain the result shown. So I "accept" these results. I know that there exists an orthog basis, because G-S tells us you can always do it. I don't think the result is unique, just as it is not unique in E^3 , because you can do an arbitrary "rotation" on one $\{e_i\}$ to get another. But fine. [Maybe it really is unique in this space, up to a scale factor.]

Now, I did a little private exercise. If you think about mapping x = (-1,1) to y = (0,2) to t = (0,N), you find that t = (N/2)(x+1) for your mapping, where I have in mind that x will be the argument of a true Legendre function in the limit $N \rightarrow \infty$. If I put this substitution for t into the $P_{2,N}$ that is shown on page 249 and take the limit, I find that it goes to $1/2(3x^2 - 1)$ which is precisely $P_2(x)$ in the continuous world. So author has normalized these things well.

Paradox: The equations for the a_i shown on the bottom of page 248 look to me to have the simple matrix form HA = 0, where H is the Hilbert matrix, and A = column vector $(1,a_1, a_2 \ldots a_m)$. You could then conclude that $A = H^{-1}\, 0 = 0$. But $a_0 = 1$ so this is wrong. That is my paradox.

Paradox resolved. There are m+1 terms in equation, but only m equations, so the matrix is not square. It has m+1 columns and m rows. The leftmost m columns of this matrix do form the m x m Hilbert matrix, it is true, as author says, it is "involved".

So at this point, we have found a nice little Legendre-like discrete orthogonal basis for our space, and the functions of t are shown on page 249.

21.27. Recall from Stakgold p 125 that, when you have an orthonormal basis $\{\mathbf{e_i}\}$, the coefficients of the "projection" of a vector **y** on a subspace is given by the "Fourier coefficient" $c_i = \langle \mathbf{e_i}, \mathbf{y} \rangle$. If things are not normalized, you get an extra $\| \mathbf{e_i} \|^2 = \langle \mathbf{e_i}, \mathbf{e_i} \rangle$ in the denominator. Now in our present problem, we have a non-normalized set of basis "functions" (functionals I guess you would say, where domain is integers 0 through N). The Fourier coefficients are shown in this problem, trivially derived, and they match the general theory. Once again, orthogonality is "the way to go". // Author Scheid now states what I just said on page 250 top, we are just applying the general theory.

What has happened here? We have solved our entire system of equations for the general least-squares discrete best fit problem! We no longer have a Hermite matrix to worry about. The exact solutions for the a_k are shown bottom of page 249 and you get these from your N+1 data points y_0 through y_N which are the values of our y(t) at t = 0,1,2...N.

The rearrangement issue: look at our final result for a_k . What happens if you increase the degree of you fitting polynomial from m to m+1 (keeping the number of points to be fit N fixed). The previous a_k

do not change at all (they are given by the same formula), and you simply have to compute one more a_k. This is the stability issue vaguely discussed in other places. You therefore have a converging solution I think is the conclusion! This is a characteristic of the ortho approach.

21.28. Here we get a formula for the size of the best-fit difference sum (the least-squares sum). Remember that we are fitting N+1 points of data with polynomials up to degree m. When m = N, the second term cancels the first, and the error goes to 0, as we would expect. Also, as we gradually increase m from 1 to m, we see the second term monotonically growing and the error monotonically reducing until it reaches 0. We can compute exactly how this happens!

21.29. So here we apply the above a_i formula to fit some data. We have N = 20. Author took a little cubic formula and added some random noise to it. He computes the first 5 a_i and we see our fitting polynomial expressed as a linear comb of the orthog functions. The table on page 251 shows the fit for m = 1,2...5. Interestingly, the best fit in terms of RMS error is m = 2. I think more comments are needed on this. The noise somehow makes the higher order fits worse. Shades of an asymptotic series.

CONTINUOUS DATA, THE LEAST-SQUARES POLYNOMIAL

Now we return to more familiar territory!

21.30 Once again, the theory machine is applied. We now have a continuous domain -1 to 1 and our Fourier coefficient is shown as a_k formula near page bottom. [This continuous domain replaces the domain t = 0,1...N of the previous section.]

21.31 Show best linear fit to $y(t) = t^2$ in (0,1) . The error is proportional to P_2 since it is the missing piece! The answer is p(t) = t - 1/6, graph is shown. One step needed here is to change variable to x so we get the required range (-1,1) in x, where we can apply our theory. Graph of result is shown.

21.32 Find best parabolic fit to sin(t) on 0 to π. This and previous both have nice graphs drawn.

21.33 "**Shifted" Legendre polys** just means that you change from normal argument to t = (1-x)/2 so things are orthogonal on (0,1). Big deal! These appear on page 774 of A&S as $P^*_n(t)$. The range (0,1) and the weight 1 and the norm factor completely define these things!

21.34 A final problem: some visual experimental data is shown, we are supposed to do a straight line fit. We need the two integrals shown [which are from the general a_k solution formula, but using those shifted Legendres], and we estimate each using Simpson's Rule on page 108 and the result is then plotted. We could have done this same problem with our discrete methods above. I don't think the answer would be exactly the same, however. Each world will have a unique answer, though.

CONTINUOUS DATA, A GENERALIZED TREATMENT

21.35 We simply modify the general theory presented earlier by adding non-negative weight w(x) into the inner product. The range (a,b) and the weight w(x) will determine the polys here called $Q_k(x)$ up to a norm constant.

21.36 Author makes my above comment about "rearrangement". The earlier terms don't change when you decide to go to next higher level poly, from m to m+1 say, due to the ortho fact.

Next few problems do the minimum of I expression, same derivation as in discrete world; then **Bessel's inequality**, which just says the magnitude of a projection of **y** is ≤ the magnitude of a vector **y**. Then 21.40 says that if your vector space is complete, then your poly converges to any function and the error approaches 0.

THE CHEBYSHEV POLYNOMIALS

These are the ones you get if you pick (-1,1) and $w(x) = 1/\sqrt{1 - x^2}$. We can compute the Fourier coefficient a_k using the general theory.

21.47 shows that $T_n(x)$ has n zeros inside (-1,1) and has mag 1 at both ends, and is +1 at the right end. No zeros outside. Inside, the magnitude is bounded by +1 and -1, hence the **equal ripple property**. These functions certainly do look useful.

21.48 If you go out to m-1 terms for your poly *and* things converge quickly, your error is roughly the mth term which you did not include. This term is $a_m T_m(x)$ and of course varies with x, BUT the variation is sort of uniform across the interval, you don't get a huge error at one end and none at the other, etc. As you move across the interval, the error is bounded in the range $(-a_m, a_m)$. This is a nice feature of the fats.

21.49 Here we use the fat polys to linearly fit our $y = t^2$ parabola again on (0,1) . The fit is computed first by the crank, and second using the little table of powers expanded in T's. The fit is a little nicer than the one we got with the Legendres in that the three max error points are now all the same in magnitude. [So I guess I need to say that the least squares fit is unique for each w(x). This is because w(x) weights the thing that you are minimizing.]

21.50 Now we try to fit sin(x) with fat polys, but author claims we cannot do the integrals! These integrals are of the weight function times sin(x) times a fat poly (or a power if you like). Just doing the power integral gives a messy F_{21} hypergeometric mess (Maple says) which author does not want to mess with. So he does a Taylor on sin(x) first, replaces each power with a combination of T(x)'s, then truncates to get a cubic fit. The graph shows how the ERROR of this fit looks compared to the error of a simple Taylor cubic fit. As expected, the Taylor is great around its expansion point (x=0), but goes to pot as you go to the ends of the interval (-1,1). In contrast, the Chubby fit has that nice uniform error behavior.

21.51. Here we change back to the discrete world for a while and we consider the set $\{ x_i \}$ of values of x in the range (-1,1) which are the zeros of $T_N(x)$ [as if we were in the Gaussian quadrature discussion.] All the discrete work we did earlier in this chapter was with equally spaced x_i , so this is something new. We find that the $T_n(x_i)$ are orthogonal if you SUM them over the x_i and there is no ugly weight function. I don't think we saw anything like this with the Legendre polys, for example. So we get this odd thing that the $T_n(x)$ are orthog if you **integrate** with weight, or if you **sum** without weight. But wait. I think if you go do the GQ again with "the general theory", you do find that all polys have a sum formula like this. I thought I did that general case, but I don't see the notes in this file. Yes, in the Hildebrand notes in the Jim section I have comments on this. But I am still unsure on the discrete sum. Maybe it is a fat special.

21.52. Still in the discrete world, we are now trying to do a best fit to some y(x) but we are only looking at the special zero points. Same theory, and we get the usual type formulas for a_k . The claim is that in this case, when m reaches N-1, your error goes to 0, again, as expected, but the error is not shown here.

21.53. Now we see why we detoured above into the discrete world. Consider again trying to fit $y = t^2$ with a straight line in a continuous sense. We set N=2 and there are two x_i and we compute the a_k using the discrete problem above. We get the same straight line fit we got before! Notice that we are sort of "continuing" p(x) [which is the lincomb of $T_k(x_i)$] between the x_i here. So this is a Gaussian Quadrature Like method of trying to fit a function with a polynomial. However, the coefficients here change as you change N.

21.54. Now we try fitting $y = t^3$ in the same way (with a line). The result is found for general N, and for N = 2, and then is also found by the continuous-x approach (where you minimize a different square). In all cases, you get the same line that we got before when doing this. Some strange sums appear here.

21.55. Now we try fitting y = | t | with a line. In this case, for any N we find that $a_1 = 0$ and that a_0 = a sinc function thing that settles to a constant value for large N. At this limit, the fit is a horizontal line at $2/\pi$. This is the same result you get with the continuous-x method.

21.56. I skip this, we are just using the fat poly theory to redo a fit we did earlier, result is the same.

This was a very long a complicated chapter! I just went and reread the "introduction" section, which really summarizes everything, and it is very good. The big tool in this chapter is the idea of using Hilbert space theory to find a best-fit. We did the discrete fit first, then continuous fit, then ended up with some discrete fits in the special Chebyshev case. Everything is laid out well in the introduction.

****

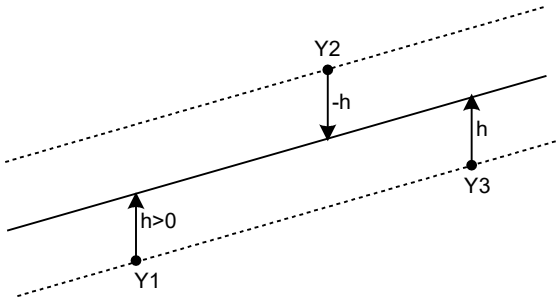
Chapter 22: Min-max polynomial approximation.

In the min-max world, we seek to "minimize the maximum of the error" | y(x) - p(x) | over a set of discrete points $x = x_i$, or over a continuous range. Here y(x) is some data and p(x) is our polynomial fit to the data. This is the L_1 norm, whereas in Chapter 21 we did the L_2 norm where we minimize the integral or sum of the errors squared at the various x. The L_2 fit could in theory have a huge max if it does not contribute much to the integral or sum. In this chapter, as in the last, we start out with discrete data. The opening discussion of the chapter is quite clear (now), and you learn why the Fat Polys are associated with min-max.

So, suppose you want to "fit" some data. Should you do a least-squares fit (Chapter 21) , or should you do a min-max fit (Chapter 22)? Or some other fit like a collocation one? I guess it depends on what kind of error you are trying to minimize. If something is going to blow up if you exceed some range, then maybe min-max is what you need. But maybe you are trying to minimize some kind of "noise" in terms of RMS power, and that would be least-squares. What should you do in statistics to fit some survey data? Maybe Scheid will comment on this before the book ends.

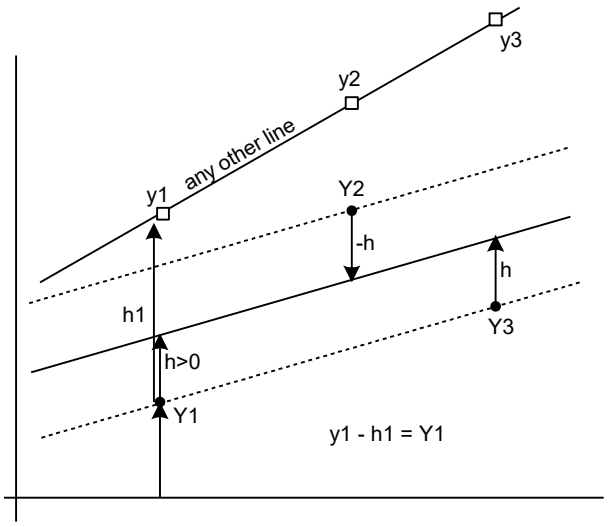
THE MIN-MAX LINE FOR DISCRETE DATA

Problems 22.1I Here is a picture



We number the points left to right 1,2,3 and they have upper case Y_i. The points in the solid line are lower case y_i. Then we connect 1 and 3 with a line. Then we draw a parallel line through point 2, be it above, below, or on our original line. The **Chebyshev line** lies half way in between. It is then easy to compute the little h heights (all same in mag) to be as given in the formula on page 269. To do this, you know M for all three lines from the top one. Compute B1 for the top one, then B2 for the bottom one, then 2h = B1 = B2 and you are done.

22.3 Now let's consider "any other line",



Here the lower case y_i are the three points on this other line at x_i . If we define H = max(h1,h2,h3), then we want to show that H > h. In the last equation in this problem, we write h as a function of the three h's of the any other line. The *largest* that the RHS can be occurs when we happen to have h_1, h_2, h_3 = H,-H,H, in which case the RHS takes the value H. For any other h_i, the RHS is smaller than this. But the RHS is equal to h, so h ≤ H. QED. We have shown that any other line but the Cheby line has worse max error, so the Cheby line must be the line of minimum max error. That is, the Cheby line is the min-max line, for the three points.

22.4 I think this is obvious: the min-max line is unique.

22.5 In **the exchange method**, you start with any three points of N points and make the Cheb line. You then calculate the h_i for all points and take note of the worst which is H in mag. If H > h, then make a

new triplet by including this point and by deleting one of the three previous points in such a way that your new triple has h-alternation in sign. Author claims it is obvious that you can do this (but not obvious to me now). For the new triple, we have h* and we know that h* > h, because we prove it in 22.6. Now we compute all the errors for this new line and compare the worst error with h*. If larger, we have to keep going. Eventually, the iteration stops, as proven below.

22.6. Here we show that in each iteration, h increases monotonically. Since there are a finite number of triples, you cannot iterate forever. You will never repeat an old triple because it will have a smaller h than your current h. Each iteration you eliminate one triple. So you have to get done eventually.

22.7 Prove that, for N points, our linear min-max fit (y = Mx + B) is the same as the line that you end up with doing the exchange method.

22.8 Example: apply the exchange data to fit a set of 20 data points. Result is shown on page 272.

OK, enough on this subject.

THE MIN-MAX LINE POLYNOMIAL FOR DISCRETE DATA

22.9 In the previous section, we dealt only with linear fits, the Chebyshev Line. In our exchange method, we picked 3 points and iterated as needed. This whole idea generalizes in a trivial manner. In the linear case, we picked 3 points and solved for our 2 free parameters (M and B) to get h, -h, h errors. In the quadratic case, you pick 4 points and solve for 3 free parameters (here called a,b,c) to get h,-h, h,-h. You then simply "do" the exchange method until it stops. That is, if you find some $h_i > h$, then you include that point in your set and throw out some other one such that you retain the alternating signs.

The interesting thing is this: if you fit with polynomial of degree m, you get m+2 points where the error is maximal. In our line (degree 1), we had 3 maximal error points. All other errors are inside the tube, so to speak, so have smaller errors.

Scheid does not pursue the general N case very far, but we see how it plays out.

CONTINUOUS DATA: THE WEIERSTRASS THEOREM

I am skipping the proof, but here is what this theorem says. If you have some continuous y(x) on (a,b), you can fit it with a polynomial p(x) such that the min-max error will be as small an ε as you want. Of course as you lower ε, the degree of p(x) will have to increase. What we are really saying is that there is a sequence of polynomials $p_n(x)$ which converges to y(x) *uniformly* on the interval (a,b). In Scheid's proof he actually constructs the sequence of polys that does the job, and these polys are known as "the Bernstein polynomials for y(x)". The proof has a few lemmas

CONTINUOUS DATA: THE CHEBYSHEV THEORY

22.17. Here we prove that, for a given degree n, we can find a polynomial P(x) that is the min-max polynomial for function y(x) on interval (a,b). The proof has about 50 steps but seems followable. I will skip it, however. The solution is not constructed in this proof.

22.18. Here we show that, if the max abs error is E, there must be some point x_1 in the range where the actual error E occurs, and there must be some point x_2 where - E occurs. You might think at first that only one of these two would have to occur, but here is the trick: suppose E occurs but -E does not. Then you could simply raise P(x) some amount and end up with a *better* min-max polynomial with E-δ . But you already have the best P(x). In other words, the best P(x) is going to be vertically centered so we hit the -E and E points each at least once.

22.19 and 22.20. Here author proves two facts that we saw in the discrete case with 3 points. For a linear fit, we saw there that we had (h,-h) max error points, but we also had a third point h with (h,-h, h) alternation. Seems pretty clear for a line and 3 points which 3 points determine the min-max best line. The exact same result holds for the continuous data: not only do we hit E and -E as noted in the previous problem, but we hit E again so we have E, -E, E. This is certainly less obvious than the 3-point situation! You sort of try out a line that gives E,-E and you rotate it until the "far end" gets down to E.

So what is proved here is that you get a third point with E, and that they alternate.

22.21 Now we generalize to say this: the min-max P(x) of degree n will be such that you get E,-E,E,-E... alternating in sign at n+2 points in the interval!

22.22 says that the min-max P(x) is unique, given the interval and given y(x).

22.23 then turns the tables around! It shows that i*f you can find* (by whatever method you like) a polynomial of degree n that has n+2 of these alternating E error points, then that must BE the P(x) min-max poly.

CONTINUOUS DATA: EXAMPLES OF MIN-MAX POLYNOMIALS

22.24. Consider $y(x) = x^{n+1}$ and we want to find a min-max p(x) of degree n. We know how to express this power as a sum of $T_i(x)$ functions from the last chapter. If we throw out the last term, we obviously are making an approximation. The error is this last term. But since this last term is $T_{n+1}(x)$, we know it hits +1 and -1 in alternating fashion n+2 times. By problem 22.23, this must BE the unique min-max poly! So you can see how the Chebyshev polys get into this min-max theory because they have that spiffy equal-error alternation property! We then make a little table for various powers. It turns out that for x^{n+1}, the min-max poly is of degree n-1 and not degree n as you might think, because of the fact that in the expansion of a power, you get only even or only odd T_i .

22.25. In the previous problem we considered specific sums of $T_i(x)$, but here consider an *arbitrary* linear combination. Each partial sum is the min-max for the sum with one more term. This is basically what we showed in the previous problem.

22.26. Here we start with a 5th order poly that is the first 3 terms of sin(x), same as in an earlier problem. We can write this in terms of T_i and we get i = 1,3 and 5 terms. If we throw out the T_5 term, we know that what is left is the min-max for when the T_5 term is included. But that is our original poly. Thus, by throwing out the i=5 term, we have constructed the min-max poly for our original poly. The result is NOT just the x and x^3 term of the original poly. It is a special poly of degree 3 which is the min-max of the original poly. This idea is called **economizing** a 5th degree poly to a 3rd degree one.

22.27 If you function y(x) is cupping up throughout (a,b), it is easy to find the n = 1 min-max line. In the triple h, -h, h the first and third will be at a and b, and the middle one is at a place where the slope equals the slope of the lines joining the endpoints. (Rolle like)

22.28 This is just an example of using the cupping up analysis in the previous problem.

22.29 Here we first *assume* that $x^2 + 1/8$ is the min-max for |x| of degree 3 (or less). We plot and discover that the errors do the right alternation thing, there are 5 points. Thus, our assumption must have been OK.

22.30. **Exchange Method in continuous world**. This is an approximation method, not a complete solution. Pick three points, maybe two at the ends and one in the middle, of your y(x) curve and fit them with a line that does the usual h, -h, h thing. Compute errors at some mesh along the interval. If you find some h* that is larger than h, slide one of your points so it hits at this point, then do another line fit. You are never done to all decimal places, but you can get however much accuracy you need by doing some exchanges! So we are using the 3-point theory and applying it to the continuous-curve world.

22.31 Here the exchange method is applied to $y(x) = e^x$ on (-1,1), but we want a n=2 min-max fit. We start with a reasonable initial quadruple and adjust it a few times with exchanges. THE END!

****

Chapter 23: Approximation by Rational Functions.

In the previous two chapters, we learned how to "fit" some y(x) with a *polynomial* to minimize error in two different senses (least-squares, min-max). In many earlier chapters, we have done other "fits" or approximations, such as Taylor series polynomial approx to a function, or collocation polynomials and such things.

All this is fine unless you have a *pole* in your function near the interval of interest. As we know, polynomials don't model poles very well. However, a rational function has zeros / poles and can in fact model poles just fine. So the subject of this chapter is doing a fit with a rational function.

In particular, just working with rational = quadratic/quadratic is pretty good for most practical problems. This is the hypothesis of problem 23.5. After a long derivation, it is shown that you can write quad/quad in a very strange-looking continued fraction. Given a table of data (near a pole, say) that you want to "interpolate" with a "fit", what you do is write out the "reciprocal differences" in a tree, and then you write down your interpolated value by inspection using the continued fraction formula.

The continued fraction has the property that it sort of converges to the answer, so "more distant points" have less effect. For example, in the little tan(x) interpolation example, the effect of the data point with x_4 makes no difference at all to 3 places. But I don't think this convergence property was really proved in this chapter. (More like the order in which we put the points?)

So that is the first topic of this chapter: doing quad/quad fits to data using the continued fraction.

A second topic is the idea of using a rational function in min-max theory! In Chapter 22, we did min-max only with polynomials, but all that stuff can be extended to rationals. The same equal-error stuff still applies. We can even do the exchange method with a rational, and some examples are given.

CONTINUED FRACTIONS AND RECIPROCAL DIFFERENCES

Problem 23.5.

Preamble: Suppose we decide we want to fit y(x) as y(x) = Mx + B. Fine, we have two unknown constants, and we can find these by fitting two points. The connection between M and B and the values x_1, y_1 , x_2, y_2 is not obvious. But, suppose we write it in this way

$$y(x) = (y_2-y_1)/(x_2- x_1)\, x \; + \; (y_1x_2 - y_2x_1)/(x_2- x_1)$$

Then we just "feed in" our data points (x_1, y_1) and (x_2, y_2) and we have our answer for y(x).

Main act: Now suppose we decide to fit y(x) = quadratic / quadratic as shown and we have the 6 coefficients shown, but really there are only 5 because it is a ratio. This is like y = Mx + B with those constants you don't know much about. However, it turns out you can write it as shown on the bottom of page 286. Here, we don't have any unknown constants like a_1 , we have only the 5 coordinates! Now it is true that the "**reciprocal differences**" ρ look pretty messy, but you compute them in a little table just the way you do for ordinary "differences" like Δ^2 and Δ^3 . Example is on page 287.

The point is that you can just insert your coordinates and write out this "continued fraction" form, then simplify it and you get your quad/quad form. Problem 23.7 does an example. Here we take a trivial y(x) [that is exact as quad/quad] and we pretend we don't know this function, we just have the 6 coordinates shown in the table. that is all the info we have and we want to fit this data to quad/quad. We write down the coords, compute the various "reciprocal differences" and then we write out the continued fraction (by inspection), and we are done! Of course here the fit is exact. You could have Excel compute the differences, for example.

Problem 23.8 Now that we have done the above "test problem" with its exact result, suppose we now try y(x) = tan(x) which is *not* an exact quad/quad. We pick a little range that is close to the pole at $\pi/2$ = 1.571 in tan(x), and we write down 5 coordinate pairs as shown on page 288. These are computed exactly from the tan(x) function of course. Then we compute the differences , and then by inspection we write out the continued fraction. If we put in a general x and simplified, we would get the quad/quad fit. But in this problem, we only care about x = 1.565. We can tell from the table that tan(x) lies between 92 and 1255. The quad/quad fit gives us tanx = 172.552 which is perfect. Interpolations of earlier chapters like Newton forward give horrible results, because they cannot model the pole!

MIN-MAX WITH RATIONAL FUNCTIONS

23.11. Here we take a single-pole rational function and a triplet of points and we find a way to make it do the h, -h, h trick. Just brute force, it can be done. In the next problem, we have an example.

23.13 The exchange method is still viable! You are trying to fit y(x) with 1/(a+bx) for example. Pick three points and do the h, -h, h trick. Compute other errors, if out of range, replace one of your points with it but maintain alternation. In other words, there was nothing about the exchange method that required that the approximating function be a polynomial. You can do the exchange idea with any class of functions you want. In the last problem of this chapter, we use the rational fit as "smoothed data", with the idea that it can removed noise from data (that the author intentionally added).

****

Chapter 24: Trigonometric Approximation

For this chapter, I first did the problems, then read the introduction, and that is what the reader should do now to get an overview of this chapter.

Scheid first does discrete-x Fourier series analysis, then continuous-x Fourier series analysis. In either case, you need to make your y(x) periodic by either even or odd continuation from some starting interval. He does not mention anything about the Fourier integral which allows you to work with non-periodic functions.

In the discrete world, the expansion is a collocation fit to N+1 values because the expansions match your data at y(n) where n = 1, 2..N. That is, these values y(n) are all that appear in the projection formulas. You then can continue your expansion off these integers to y(x), and that is why you have a collocation fit. It is not a collocation polynomial, however, because it uses trig functions.

A major result is that if you truncate either of these series expansions, the fit you get is a least-squares fit. This is a characteristic of working with orthogonal basis functions and we saw it before with the $T_n(x)$ Chebyshev expansion, for example (and you do not see it in a simple power series expansion). If you don't truncate, you collocate through all the points (discrete-x case). If you do truncate, you are going to miss the points a bit, but you get least squares fit to the non-truncated collocation result. Truncation is a nice way to smooth data and also to work with derivatives.

You can apply a blurring method to a Fourier series to make it converge faster, and this is pretty much a simple box filter theory application and results in a sinc function dropping off as you go out in the terms and this is why things converge faster. The name associated with this is Lanczos Sigma factors. Remember this book was written in 1968 perhaps before filter theory was a lingua franca.

TRIG SUMS BY COLLOCATION

24.1 through 24.5 For the discrete world, basis functions are $f_j(k) = \sin[2\pi jk/(N+1)]$ *and* $\cos[2\pi jk/(N+1)]$ for j = 0 to N. The discrete variable here is k while the basis function label is j, and the functions are symmetrical in k and j. For sin, j=0 does not give a useful function, so we have only j=1 to N or N functions. For cos we include the constant, so there are 2N+1 basis functions it would seem. *You would think* therefore that the general expansion formula would be this:

$$y(n) = \sum_{k=0}^{N} \{ A_k \cos[2\pi nk/(N+1)] + B_k \sin[2\pi nk/(N+1)] \}$$

where B_0 does not really enter the fray. It is useful to write this as

$$y(n) = \sum_{k=0}^{N} \{ A_k \cos[\alpha_n k] + B_k \sin[\alpha_n k] \} \qquad \text{where } \alpha_n = 2\pi * n / (N+1)$$

To determine the projections, you use orthogonality and conclude that , for example,

$$A_\ell = 2/(N+1) * \sum_{n=0}^{N} y(n) * \cos[\alpha_n \ell] \qquad \ell \neq 0$$

So if I give you a set of N+1 values y(n), you can compute the A_k and B_k and then the expansion shown above will be a **collocation formula** if we "continue" n to general real x. We will have a fit called y(x) which will match our N+1 values at the integers x = 0 to N. This is a trig fit, not a poly fit!

But there is something wrong with this argument. If true, we are requiring 2N-1 parameters (the A_k and the B_k) to get a fit to our N+1 data points. It must be true that each A_ℓ for $\ell > N/2$ (N even) is equal to some B_ℓ for $\ell \le N/2$, and those such higher index A_ℓ's are really not needed. The same would be true for the higher B_ℓ coefficients. I think you show could show this using formulas like $\sin(x) = \cos(x + K\pi/2)$ or what have you, but I have not done this. Scheid has figured this all out, and you have to separate into the even and odd N cases for your expansions. For N even, the expansion is then this:

$$y(n) = \sum_{k=0}^{N/2} \{ A_k \cos[2\pi nk/(N+1)] + B_k \sin[2\pi nk/(N+1)] \}$$

rather than a sum all the way up to N. Then you end up with N+1 coefficients total representing your N+1 collocation points y(n). So I accept all his results.

Scheid fakes the proof of orthogonality by arm-waving about "equally spaced". The real proof I have shown in pencil, but it requires complex numbers and that is why Scheid avoids it. I think this entire book avoids complex numbers, that was one of his requirements perhaps.

In 24.3, he directly shows that the projections as described by him do exactly reproduce the expansion. He uses N = 2L = even.

In problem 24.5 he quotes the expansion and projections for N = odd, and things are in fact a little different. This one, however, he does not "prove" as he did the even case.

Comments: I am sure all this stuff is much cleaner if presented in complex notation. You could have y(n) be complex values and expand on exponentials with complex coefficients C = A + iB with A and B real, and so on.

TRIG SUMS BY LEAST SQUARES.

24.6 Suppose you have a collocation expansion for y(x) as outlined in the previous section. Suppose you try to "best fit" this thing in terms of least-squares with a similar expansion but of fewer terms. That is to say, in your fit, you run your expansion sum not all the way up to L = N/2, but to some M < L, and with coefficients that are complete general. What he shows is that the fitting expansion is merely a truncation of the full expansion with the same coefficients. This is typical of what happens in orthogonality world. This is again a special case of our general theory.

EVEN OR ODD PERIODIC FUNCTIONS

24.8 Up to this point, we never said that y(n) was periodic. We did our collocation fit to N+1 points y(n). I think the resulting expansion of course IS periodic, but all we wanted was a collocation. Now here we explicitly assume that y(n) is periodic. This works well with N= 2L - 1= odd, so N+1 = even = 2L = P the period, so the period is a nice even number of values. The formulas seem right, and of course you are allowed to shift the summation index now since y(n) is assumed periodic.

25.9 If y(n) is odd, then the a_i vanish and we get a folded formula for the b_i. In 25.10 we do an example of an odd function.

24.11. Here are the corresponding results for an even function y(n). Then an example.

CONTINUOUS DATA. THE FOURIER SERIES

We proceed in parallel fashion to the previous section. We first write the orthogonalities in 24.13 and now we have the two sets of basis functions as sin(jt) where t is a continuous variable and j is an integer basis function label that goes j = 0 to ∞. Then in 24.14 we write the projection formulas and the fourier series expansion. The expansion is still a series, it is the projections that are now integrals. In 24.15 we do an example y(t) = | t | made periodic. Then in 24.16 we do a discontinuous example and we find that it does not converge very fast, as we would expect. Then in 24.17 we fit to something that already pretty much is a sin, and we find a very convergent series, again as you would expect.

24.18. Here is a problem I bumped into earlier in the book, problem 17.30. He writes a Fourier series expansion for the **Bernoulli** function $B_n(x)$ extended to be periodic. His proof is to do the first few terms and then claim the general result follows by induction. I think this is where he has missed a factor of n!, but I don't see how that happens because I have not followed the details here.

24.19 This is a good one. We know from the discrete theory that we can replace the N+1 y(n) points with N+1 coefficients, and these are obtained from projection formulas which involve sums of y(n). As we let N get very large, the sum has many terms, and it is in fact the trapezoidal rule approximation to the continuous Fourier coefficient that you could get from an infinity of points. It has to work this way!

LEAST SQUARES, CONTINUOUS DATA

24.20. We learn once again that if you truncate a Fourier series expansion, that is the best least squares fit to the order to the full expansion, the same old orthog general result.

SMOOTHING BY FOURIER ANALYSIS

24.22 This is sort of JPEG. If you have some smooth data with white noise, say, if you throw away higher order Fourier coefficients, you will be discarding mostly noise. So truncation is a way to smooth the data. Of course in doing so, you do retain the lower frequency part of the noise. Problem 24.23 is an example of smoothing in this way. Then problem 24.24 uses the idea that smoothed data ought to give you a better model for the derivative. I have not really studied this problem, but get the gist.

THE LANCZOS SIGMA FACTOR

24.25 An interesting idea. Take a truncated Fourier Series expansion and just blur it. Do this by integrating both sides of the expansion equation over a symmetric range of width $2\pi/n$, where n-1 refers to the last term in the truncated series. If you do this, you get a new Fourier expansion which has the blurred function on the left, here called s(t), and the right side is the same as before, with the addition of blurring sinc functions which are called the **Lanczos sigma factors** (because some guy of that name called these sinc functions σ factors.). The claim is that the presence of these sinc functions causes the series to converge faster!

24.26. Here we do an example. We write a truncated Fourier series for a square wave (n = 26), and we note that the normal Fourier expansion has very slow log style convergence 1/K integers. The blurred version with the extra sinc factors is then compared to the original. You see that the sinc function is "1" in the first term, and by the time you have reached the 25th term you have fallen down from the central peak of the sync function (not sure how far, he might have plotted it). This dropping off of the sinc function as you move up the terms is the improvement in convergence! And graphically, you see that the non-blurred y(t) expansion has ringing, whereas the blurred one has this ringing filtered away. I think all we are saying is that we are running the square wave through a low pass box filter.

24.27. Here we do a derivative case again, and the point here is that the sinc filter trick makes a series for the derivative at least converge! Without the filter, it diverges! I then skip the last problem altogether.

****

Chapter 25: Nonlinear Algebra

The first subject is finding the roots of an arbitrary equation f(x) = 0. The second is solving a system of equations like f(x,y) = 0 and g(x,y)= 0. When f(x) = 0 has complex roots, you need extra work and extra methods. Newton-Raphson is the main act in this chapter, giving fast quadratic convergence to solutions of the 1 and 2 dim problems just mentioned. Other methods get you to approximate starting points.

Some methods apply to general functions f(x), others apply only to polynomials p(x). I have marked these at the start of each section. Some methods try to find all roots at once, others only one.

Historically, the oldest method mentioned here is the regula falsi method of finding roots of a function. One web site claims Greeks in 1850 BC. Next in line would be Newton's method of 1671, see below. Others were near that era. Sadly, Scheid seems uninterested in history.

This is a long and fairly brutal chapter with many methods mentioned.

Maple (fsolve) of course can do all this stuff, but things are proprietary I suspect. I will be it uses many of the methods outlined below. I wonder how hard it is to find software to do things that is already tested. Surely MATLAB has lots of stuff in it as well (such as roots()).

THE ITERATIVE METHOD //general

Problem 25.1 discusses this idea where you recast the equation f(x) = 0 into the form **F(x) = x**, and there are usually several ways to do this. You have to make sure that F '(x) < 1 in the interval around where you suspect there is a solution (a "root"). The upper bound on F'(x) is called L and so L < 1 is required. Then if you simply iterate $x_n = F(x_{n-1})$, you will spiral into the solution after many iterations. The spiral-in idea is shown on page 313. Here is my explanation; you start with some x_0 and then you compute $F(x_0)$ and put a fat dot there in the picture. You then find the value of x such that $x = F(x_0)$. So if $F(x_0) = 5$, you then go over to x = 5. This means you move from your first fat dot to the 45 degree line because you are setting x = y and this x will be x_1 , and the y value is $F(x_0)$. Then you go vertically until you hit the curve F(x) and that value will be $F(x_1)$ and you put another fat dot there. Because the slope is < 1, these dots converge onto the root. Other picture shows how you get a diverging spiral if this is not the case. This method seems to have no formal name other than **the iterative method**. No reference.

In 25.2 we apply this to a certain f(x) = 0 that some Leonardo guy worked with in 1225. We pick a form of x = F(x) with the right slope requirement, and after 24 iterations we have the same number that Leo got.

In 25.3 we learn that in this method, the **error convergence** behaves as roughly $e_n = F'(r)\, e_{n-1}$ so you are only decreasing your error by the slope which is L. If this is .0001, great, but if it is -.44 as in our problem, convergence is slow.

In 25.4 we do a little trick. Once you have three iterative values, you can compute a fourth that is more accurate than just doing the 4th iteration, because it takes into account the quadratic term in the Taylor expansion which we have so far ignored. For some reason, this little method is called the **Aitken's Process**.

In 25.5 we do the first 12 iterations of the 24 in our first problem, then apply the trick, and boom, we are at the same accuracy doing all 24 iterations. So this is a **convergence accelerator**.

In 25.6 we use the Aitken idea on every 4th iteration, and now we only need 9 iterations to get to Leonardo's accuracy. This systematic plan of doing the Aitken's trick every fourth time is called the **Stephensen's Method**.

In 25.7 he shows that if you pick the wrong form for F(x) = x, you can diverge.

THE NEWTON METHOD // general

*This method is also called the **Newton-Raphson** method. Newton discovered it in 1671 and wrote it up in his Method of Fluxions (Latin), which was circulated to math people, but the English translation came out in 1736, after Newton's death. Joseph Raphson published the method in 1691, he was a colleague of Newton's, so probably we should credit Newton as the real discoverer. This is a period of science and math that I should study some day, I probably already have a book on the subject.*

In 25.8 we write f(x) as Taylor and just keep the first slope term and this gives the Newton iteration formula which is $x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1})$.

25.9 **Graphically** on page 316 we see a piece of our function f(x) coming down to the x axis, the intersection being the root f(r) = 0. We start at some x_{n-1} and we pretend that we have a straight line at this point and we use that to guess where the root is, and we hit the axis at point x_n. A good guess, but we missed, so we do it again, going up from this zero point to the curve again and starting there. We just sawtooth our way in, it looks bulletproof to me! No slope requirements. Of course you have to know that you are on a smooth isolated region of a root, ie, you need a monotonic segment I think.

In 25.10 we apply this to Leo's problem and we get the same answer after a mere **4 iterations**!

In 25.11 we see that we have **quadratic convergence** in this method because e_n is proportional to e_{n-1}^2. Thus if you have e = .001, your next e will be .00001. This means you get about 2 decimal places per iteration which is pretty cool. This is all new to me today, but I guess I saw it somewhere in my life.

In 25.12 we get a trivial iterative formula for doing **square roots**, and in the next problem we do 5 iterations to get the result to 10 decimal places starting stupidly at 1.

In 15.14 we have the same idea for **p^{th} root** and then we use it for cube root of 2.

So when you are running a program and you do sqrt(7.223), it is probably calling a little routine that does just this! In fact my 1980 spiral book on this subject says Newton is used just this way, and is called Heron's Formula. I am amazed that I have been ignorant of such important and simple math methods!

INTERPOLATION METHODS // general

25.16. This method is called **regula falsi**, which means "false rule", but I don't see the grammar:

regula -ae f. [a ruler , plank]. Transf. [a rule, pattern, model].

falsus -a -um partic. from fallo; q.v

I think the word falsus here means "a false position" the way we would say a liar is a lying person with the word person omitted. So perhaps position is masculine or an omitted noun is assumed such, so falsi means "of false position". So we really do have "rule of false position".

In any event, you need to first identify an intersection of your curve y = f(x) with y=0 and start with points x_1 and x_2 on either side of the zero. You just imagine that the curve is a straight line between these points and compute the value of the zero x_3 for that line. In other words, you interpolate the curve by a straight line passing through two points on the curve. You then move from the computed zero x_3 out to the curve again at $y = f(x_3)$. This point x_3 will be closer to the true root than previous points. You then pick a new pair by taking x_3 along with whichever previous point lies on the opposite side of the y=0 line, and you iterate.

In 25.17 we apply this to our prototype problem of Leonardo and we find that convergence is between Newton and the iteration method.

Compare this to the Newton method: here, our estimate is a line through two points which lie on the curve. In Newton, our estimate is a line that is tangent to the curve at one point. Newton works better.

In 25.28 author comments that you could use a quadratic instead of a line. You could start with 2 points, do the above to get a third, then use your 3 points and produce a quadratic poly that passes through those 3 points. We know how to do this of course from our poly fit chapters with unequal spacing. You can solve that to see where the quadratic intersects the y=0 line and use that as x_4 and then work with triples from then on. The math looks pretty ugly, Newton I am sure is much better. No examples given.

BERNOULLI'S METHOD **// polynomial**

25.19 Here we lean on our difference equation theory (see separate document). The general solution to the homo equation is as shown for x_k where the r_i are the roots of the characteristic polynomial p(x) shown. We can write the ratio of x_{k+1}/x_k as shown with no assumptions other than $c_1 \neq 0$ and $r_1 \neq 0$. Notice that there is a factor of r_1 sitting out front. Now, assume that we order things so that r_1 is the largest root in magnitude, don't care about any ordering on the other roots. Scheid refers to this as the "dominant zero" without defining the word **dominant**. What happens to the ratio x_{k+1}/x_k as $k \to \infty$? The little r/r powers go to zero and we find that the limit is exactly r_1 . So this is a way to find the largest root r_1 !!

25.20. An example of using the Bernoulli method. Here we write down our polynomial and its "associated" difference equation. The order here is 4, so we have to start with four (standard) initial conditions on the x_k . Then we compute the ratios x_{k+1}/x_k and we see them approaching 2. So this must be the largest root (the dominant root) of the equation.

25.21 Suppose the largest in magnitude root is really complex conjugate pair. Then we call these two r_1 and r_2 and all the rest are smaller. The basis functions are still like r_1^k but we write $r_1 = r\, e^{i\phi}$ and grind through the details. We end up now with two somewhat more complicated ratios of x_k things which, in the large k limit, are supposed to approach the quantities shown. From one you can deduce r, and from the other you get ϕ , and thus you find your complex pair $r_1 = r\, e^{i\phi}$ and $r_2 = r\, e^{-i\phi}$.

25.22 Gives an example of doing this. It turns out that our friend Leonardo's equation has this situation, the largest roots are a complex pair. We find the answers.

DEFLATION **// polynomial**

The idea here is simple. Once you find a root by some method (like Bernoulli to find the largest root), you can divide your poly by $(x - r_1)$ to get a new poly of one lower degree. You **deflate** the poly of degree n to a new poly of degree n-1. This is fine in theory, but in practice if you r_1 root is off, you induce errors into all the other roots. But in theory, you could keep applying Bernoulli to the deflated poly and keep going in this manner until all roots are found.

The QD Algorithm (Quotient Difference) // polynomial

25.25 In this problem, the two quantities to be iterated are defined, here called q and d. The paper I found calls them q and e. The two little iterative equations are called **rhombus rules** because, if you arrange a table of q and d columns as shown on the top of page 322, you can compute any element from the three elements on the diamond (rhombus) to the left of your point. So you can do the double iteration and enter numbers into this little graph.

The theory is beyond the scope of this book, but the claim Scheid makes is this: as you come down the q columns, the numbers approach the roots, and as you come down the d columns, the numbers approach zero. The d columns are sort of errors and they are supposed to go to zero.

Notice that in the method as described so far, you fill out the table more or less in columns going left to right, but still you have to be coming down, so it is really a 2D fill-out. I don't really understand how you get started in that left column, because I don't quite understand applying the rhombus rule idea, but I could understand it if I really wanted to. This method is the **column by column method**.

25.26. Here we take our Fibonacci-sequence-generating poly $x^2 - x - 1$ and apply the QD method. You see the two q columns going down and seeming to approach values, while the zero d columns go to zero, but you see roundoff error building up as well.

25.27. This problem just quotes the theoretical claims already stated above.

25.28. If your poly has complex root pairs, the QD method will fail in those pairs of columns, the error d column bounces around (probably because we somehow keep missing something since it is complex). In this case, a method for finding the complex pair is given.

25.29. Here we show how to fill out the same graphical table using the row-by-row method. It is the same table I think, but you put some bogus numbers in the first two rows, and then you work down by rows, instead of across by columns. The claim is that you get better results.

25.30. Here we repeat the same Fibonacci poly example using row by row. The right most columns do seem better.

25.31. Here we take a fourth-degree poly with known roots 1,2,3,4 and try row by row. Things seem to be progressing well, if slowly. The main idea is you use a global method like this to sort of find out in what areas the zeros are located, then use Newton to zero in quadratically on the finally results.

25.32. We here apply row by row QD to Leonardo's equation and, as expected, the first two column pairs bounce around because we know these roots are a complex pair. We then apply the little trick of 25.28 to see how this complex pair is looking. Scheid remarks that you can in fact use Newton with complex math, but he has a better plan saved for later.

I found a 1964 scanned PDF "technical report" from Stanford that reviews the column and row methods in a little more detail. I will save it with these Scheid notes just for fun.

I imagine that lots of these methods are no longer used due to increased CPU power where you can just scan a function and more or less brute-force locate the roots and Newton in on them. But this history is always interesting.

STURM SEQUENCES // general and polynomial

This whole idea seems clumsy and ugly. You start with your poly of interest and call it $f_0(x)$. You construct from that by "the Euclidean algorithm" (no explanation of this phrase is given) . This means that you start with $f_1(x) = f_0'(x)$. Then you do some ugly divisions of polys to get f_2, and f_3 and so on as shown bottom of page 325. The L(x) are the quotients, and the $f_n(x)$ are the remainder functions, so degree is going down by one in each step. I think when you reach some $f_n = 1$, you are done. This then is a sequence of functions, and it is claimed that it has the strange properties of a "Sturm sequence" which are given in problem 25.33. Unfortunately, the conditions listed seem to conflict with each other, and I cannot find on the web anyone else who talks about things in this exact manner. So I will just punt and summarize the claims. You construct the sequence, and pick an (a,b) region, and find some sign changes, and you use the number of sign changes to figure out how many roots there are in (a,b). Then by playing with (a,b), you can perhaps isolate the roots to certain areas, then apply Newton to get them as accurate numbers. This whole scheme is very poorly explained I am afraid. No appropriate graphic is provided.

I think this method has something to do with Rolle's theorem, just a hunch.

SYSTEMS OF EQUATIONS, ITERATIVE METHODS // general

25.38. Here we look at finding x,y which solve f(x,y) = 0 and g(x,y) = 0 at the same time. This is a generalization of the root-finding problem f(x) = 0. Graphically, you can imagine f(x,y)= 0 describing some curve in the x,y plane, and g(x,y) = 0 is some other curve, and you are looking for the points of intersection. When f and g are low degree, as in example 25.39, you can find the exact answer without numerical work, but I suspect with higher degree you cannot.

The method is sort of a **dual Newton-Raphson** method. Derivation of the equations is done here and is very simple. Letter h is used for a small step in x, and k for a small step in y. At each level n of the dual iteration, you have to evaluate f and g with both arguments at level n-1. This gives you 2 equations in 2 unknowns for h and k at that same level, so you use Cramer's rule or some such each time to solve these for h and k. Then you advance x and y by these amounts h and k, and move to the next level. So really there are the four equations you see in this problem.

25.39. So here we apply the method to find the intersections of a circle and a hyperbola (there are 4 solution points). As usual, you assume some benign initial values for x_0 and y_0. Starting at (1,1) which is in the neighborhood of the upper right solution, it takes about 3 iterations to get pretty good results. Convergence is quadratic, as with the single variable Newton method. I bet there is some very fancy graphic you can draw which shows how you zero in on a solution. Notice that this only gives the one point of intersection that we started near.

25.40 Other methods for x,y problems? One idea is a simple generalization of the x = F(x) iteration method that we started this entire study with. You then have $x_n = F(x_{n-1}, y_{n-1})$ and same for G. An example is done for two little equations shown.

METHOD OF STEEPEST DESCENT // general

25.41. This is another way to solve the simultaneous f(x,y) = 0 and g(x,y) = 0 problem. Write S(x,y) as the sum of the abs squares of these things, then we want to minimize S. Visualize S(x,y) as a mountain terrain and you are on skis. Imagine that S(x,y) has lots of local minima bowls. You hope that one of these bowls goes all the way down to S(x,y) = 0. If it does, then you have found a solution! If you get into the wrong bowl, you ski down to a minimum that is > 0 and you have failed, you have to climb back up and hunt some more!

How do you ski down? You start at some arbitrary place, you compute the gradient ∇S and you then know that -∇S points you straight downhill. In the version of the algorithm presented, we travel along in this direction with a parameter t until S starts increasing, which happens when we pass through the horizontal contour line at some point. Then recompute the gradient, make your right angle turn, and do it again. The picture on the bottom of page 328 shows this perfectly. Another approach would be to just travel in finite steps and recompute the gradient, or perhaps to half way to the contour, etc.

25.42 Here we apply the original idea to our problem done earlier with sines and cosines. We start at a lucky good place (.5,.5), compute grad. We then compute S(t), based on travelling in this new direction, and minimize S(t) to find the contour line, and we stop there and call that (x_1,y_1). In this example, after about 5 ski shots, we end up maybe 2-3 decimal places of the solution (x,y). Convergence is slow, so you get close this way then use Newton.

25.43. This shows how you can get off into the wrong bowl if you start in a bad direction because you started at a bad starting position. You mostly have to be in the right bowl at the start!

25.44 Author claims there are MANY variations of this idea of skiing down the hill to the minimum. I could imagine this also working for f(x,y,z) and beyond. The name "steepest descent" is certainly a good one.

BAIRSTOW's METHOD using QUADRATIC FACTORS // polynomial complex pair

This is a method for finding a pair of complex roots *for a polynomial*. We start with some poly p(x) of order n. Suppose we know that it has a pair of CC roots at a ± ib. This will then cause (x - a -ib)(x - a + ib) = $(x-a)^2 - b^2 = x^2 - 2ax + (a^2 - b^2) = (x^2 - ux - v)$ to appear as a factor. This is the "quadratic factor" being referred to in the section title. If we knew the roots exactly, we would say:

$$p(x) = (x^2 - ux - v)\, q(x)$$

where q(x) is the rest of the poly with the other roots. If we start with slightly wrong u,v we will end up with something like this:

$$p(x) = (x^2 - ux - v)\, q(x) + r(x)$$

Now, suppose we start with some wrong u,v. We could in theory divide p(x) by $(x^2 - ux - v)$ and we would get some q(x) and some r(x). We know that r(x) will be of the form Ax + B since it is the remainder after division by a quadratic. Rewrite this as r(x) = C(x-u) + D. Fine.

Now, lets assume that p(x) has coefficients a_i that are given. How do we find the coefficients of q(x) and r(x)? Well, insert it all in like so into the above equation:

$$\Sigma_{i=0}^{n} a_i x^{n-i} = (x^2 - ux - v) \Sigma_{i=0}^{n-2} b_i x^{n-2-i} + C(x-u) + D$$

$$a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots a_{n-1} x + a_n = (x^2 - ux - v) * (b_0 x^{n-2} + b_1 x^{n-3} \ldots b_{n-1} x + b_{n-2}) + C(x-u) + D$$

Notice that in the sum for q(x) the coefficients are b_0 up to b_{n-2} only. There is no b_{n-1} or b_n.

Now we match the coefficients of powers on the two sides. I have fiddled with this and keep getting a mess, but here are the conclusions

(1) We get the following recursion for the b's based on the lower numbered b's

$$b_k = a_k + u\, b_{k-1} + v\, b_{k-2} \qquad k = 0 \text{ to } n$$

(2) It turns out that we can write $C = b_{n-1}$ and $D = b_n$ (thus defining two higher index b's) and the equation above then can be continued up to these new indices for the b's, and you maintain balance of the powers. This is certainly not very obvious, and I am not going to take the time to prove it.

Scheid just assumes this at the start and then gets his results. So now at least we know why we are able to write this:

$$r(x) = C(x-u) + D = b_{n-1}(x-u) + b_n$$

When first presented, this seems very strange.

Now, if we took the exact right values for u,v (based on a,b), we could have C = D = 0. But if we take wrong values, they are not zero. We then have these two functions really of u and v

$$b_{n-1}(u,v) \qquad b_n(u,v)$$

Our game is now to find values for u,v that make both these be 0, then we have our solution! This is basically a dual Newton iteration method. We define some partial derivatives of these functions called c and d, show that c = d, and then somehow we come up with a dual Newton iteration that works like so: Start with some good u_0, v_0 . Make the little table and list the given a_i across the second row. Then compute the b and c guys using the iteration formulas I have underlined in red. Once you have the first table, use the b and c data there to compute corrections h and k to the initial u and v. This gives you u_1 and v_1 which are better. Then do another iteration, making another table, and so on.

This looks very similar to the dual Newton we did earlier with some variation.

The method then is called Bairstow's Method.

25.48. Here we apply the Bairstow method. It is all pretty automatic and we watch b_2 and b_3 heading to 0. The final u and v that come out can then be converted to a and b and you have then found your complex roots using this method. The results are shown to lots of decimal places, and we get the usual quadratic convergence.

****

Chapter 26: Linear Systems // 24 pages + problems

26.1 We start out with the method of **Gaussian Elimination** to solve Ax = c. In fairly stupid fashion, you just fiddle till the system is in upper-right triangular form. It helps to get the largest number of all those available into the upper left corner (the pivot position) for each stage. You first eliminate the leftmost entry in all the rows but the first. Now the first column is all zeros except the top. Next, you do this to the next column, leaving the first row glued down. For each stage, you get the largest number up to the pivot point by just reordering the equations (swap rows), then you clear out under that point by multiply and subtract. If all works out, then you read off the answers by **back-substitution**.

Obviously this cannot work if A is a singular matrix. Something must go wrong. Here from my downloaded PDF is a problem case:

Example 6.5 *The system*

$$\begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & = & -1, \\ 2x_1 & + & 2x_2 & + & 5x_3 & = & -8, \\ 4x_1 & + & 4x_2 & + & 8x_3 & = & -14, \end{array} \tag{6.6}$$

corresponds to the augmented matrix

$$\begin{pmatrix} \boxed{1} & 1 & 1 & -1 \\ 2 & 2 & 5 & -8 \\ 4 & 4 & 8 & -14 \end{pmatrix}.$$

As before, begin by using the first pivot, 1*, to eliminate the first column:*

$$\begin{pmatrix} 1 & 1 & 1 & -1 \\ 0 & \boxed{0} & 3 & -6 \\ 0 & 0 & 4 & -10 \end{pmatrix}. \tag{6.7}$$

Now the entry in the pivot position is zero, and the situation cannot be remedied by pivoting.

In fact the system (6.6) has *no solution*; the final two equations in (6.7) read $3x_3 = -6$ and $4x_3 = -10$, and are therefore inconsistent. In general, if no nonzero pivot can be found, this corresponds to the original matrix $\boldsymbol{A}$ being singular, *i.e.* $|\boldsymbol{A}| = 0$.

As an aside, we should point out that $\boldsymbol{A}$ being singular does not necessarily imply that $\boldsymbol{Ax} = \boldsymbol{b}$ has no solution.

You see the all the candidate pivot numbers in (6.7) are zero. (We are showing the right side of AX=b as the right column of this augmented matrix, ignore it). So we have reached a point where we "cannot pivot", and that tells you the matrix A was singular.

26.2 Scheid here shows that Gaussian elimination is order n^3 in operations, pretty horrible, and causes lots of roundoff error, although I do not quite see how you estimate this error as sqrt(num ops).

26.3 Here is a great demonstration of the effect of roundoff error. Scheid pretends our computer can only manage 2 decimal places. When he repeats the previous solution with this constraint, the numbers come out miles off! He did select a **Hilbert-like matrix** just to make it really bad, the point is well made for single or double precision.

26.4 The **residual method** is a little hazy to me. It is a sort of subtraction idea. You have some roughly correct solution X to Ax = c, so lets say we have AX = c + R where R is the **residual vector** whose presence non-zero tells you of the error. Now let x = X + h to get A(X+h) = c => c+R + Ah = c which then says Ah = -R. You have replaced the original Ax=c problem with this Ah = -R problem where the numbers are hopefully relatively small. Errors in Gaussian Elimination then have less effect.

26.5 We now apply this residual method. We use as our X values the lousy solution obtained in 26.3 and we first compute R = AX-c . Then we do the usual thing on Ah = -R to get the h shown. Let's review:

exact solution	x = (9, -36, 30)	1,2,3	
lousy solution	x = (7, -23, 17)		
residual method	x = (11, -44, 37)		// done with 2-digit computer

So yes, the last process with the residuals got our lousy answer to become better.

26.6. We know this theorem well: if Ax = 0 => x = 0, then Ax = b has a unique solution. I know this of course because the condition means that A is invertible. This is called the **Fundamental Theorem of Linear Algebra**.

Well, I will skip his proof since not important to me. But we do get the idea that if you cannot find a pivot because all candidates are 0, then A is singular.

Note added by me: I think you can continue the Gauss elimination process and clear out the upper right triangle as well, and get all ones on the diagonal! Then the numbers on the right ARE the solution. This extended process is called **Gauss-Jordan Elimination**.

GAUSS-SEIDEL ITERATION AND OVER-RELAXATION

26.7 To maintain student interest, Scheid has written down a linear system in slightly different form and has applied it to a physical system of dogs in corridors. The first equation is derived as follows: dog at vertex 1 has 1/4 chance to go in any of 4 directions. If he goes up or left, he exits not on the bottom, so we have 0 + 0 for these two cases. If he goes to the right to point 2, his prob of bottom-edge exit is P_2 and if he does down it is P_4 . Notice that there are some "driving terms" in the form of 1's in the last three equations. You can of course write this whole thing as Ax = c, but leave it as is, more in the form x= Ax + b. Reminds us of writing f(x) = 0 as x = F(x).

The relaxation idea is this: just make up some starting values for x, like x all 0. Then cycle through all the equations (9 in this case) and treat them as $x_n = F(x_{n-1})$, and in this way build up the table shown. You can see that things do in fact approach stable limits in this case. For example, $P_1 = 0.71$ seems close to the perfect answer and it is equal to P_3 as the geometry requires.

So relaxation just means cycling through your system of equations with some reasonable starting point, and you "hope" things stabilize so iterations then just repeat and row stays the same.

26.8 It turns out this method works if your Ax = b has an A matrix which is symmetric and positive definite (but this is not proved here).

26.9. The above was just an example of relaxation. A formal approach is to decompose A = L + 1 + U and then iterate the vector equation shown. This formal thing is the Gauss-Seidel method, but he does not do any examples with it.

26.10. The constant in front of the square bracket is 1, but maybe if you make it 1.2 your system will relax faster to a solution.

26.11. Somehow he reformulates the dog problem in this L + 1 + U form and uses the accelerated G-S method with w = 1.2 and he makes a new table top of page 342 which shows we get to a stable result in fewer rows, 6 instead of 10. Think of the w as controlling the amount you step in each iteration. Probably too much w makes you overshoot and you have oscillation.

UNSTABLE AND ILL-CONDITIONED SYSTEMS

26.12 The general idea is that if small errors in the coefficients cause huge errors in the solution x, then the matrix in Ax = c is ill-conditioned or unstable. Norms are mentioned, but the theory of instability is not discussed. I don't think I have this subject in any of my books. I can come back to this at some later time.

26.13 Here we have a dramatic illustration of a badly behaved matrix where we make a miniscule change in one parameter and the answer moves from -100 to + 100. A nice graphical explanation in this case is given as two nearly parallel lines, and you tip them ever so slightly one way or the other and you make a huge difference on their intersection points.

26.14. Here we write $\mathbf{A}\,\delta\mathbf{x} = \delta\mathbf{c} - \delta\mathbf{A}\,\mathbf{x}$ and we can regard this I think as a detector of how small changes in A or c will affect the solution x. The second order term $\delta A \delta x$ is ignored here.

26.15. Here we consider the $\delta\mathbf{A}$ implied by the two systems shown. We compute $\delta\mathbf{x}$ by solving our equation of the last problem (with $\delta\mathbf{c} = 0$) and find that $\delta\mathbf{x}$ = very large numbers. So this confirms that this matrix **A** is relatively unstable. The only differences are between .33 and 1/3 in two places. As noted earlier, it is a Hilbert matrix, so we expect bad news.

26.16 Here instead we vary $\delta\mathbf{c}$ and keep the matrix **A** fixed. Results are simply quoted, and again, small changes $\delta\mathbf{c}$ cause large changes $\delta\mathbf{x}$, so this Wilson Matrix is unstable too.

MATRIX INVERSION BY ELIMINATION (this is just Gauss-Jordan elimination I now realize.)

26.17 The description of what to do is clear, but not why it works! Start with some Ax = c. We can represent this as the following matrix

a	a	a	c
a	a	a	c
a	a	a	c

where I just leave off the obvious subscripts. Now we can do all the usual "rules" on this thing without changing the solution of Ax = c. For example, we can take a row and multiply it by 2 and nothing changes as you can see by doing $\Sigma a_{ij}x_j = c_i$. If we double the top row of a's, that doubles the sum, but we have doubled the thing it is equal to on the right, the top c. Similarly, we can add multiples of one row to another. I am not used to this idea of doing the determinant reduction rules on this sort of extended matrix, but I think it is solid. Now suppose by applying our rules we can get this thing to look like this:

1	0	0	d
0	1	0	d
0	0	1	d

where now we have **I x** = **d** as the equation being represented, and the solution x has not changed! Well, the solution must be x = d.

Now let's go back and be more particular. Suppose we start with this

a	a	a	1
a	a	a	0
a	a	a	0

and we get to the form shown above where x = d. We started with Ax = U_1 and that means x = A^{-1} U_1. Now, suppose we write out [A^{-1}] = [C1 C2 C3] where we show the three columns of A^{-1}. Then when we write x = $A^{-1}U_1$ we get [C1 C2 C3] U_1 = C1. That is to say, we get x = C1. Thus, the d's above are the first column of the inverse matrix. We just repeat this process twice to get the C2 and C3. So why not write it all in a single extended matrix which represents these three problems all at once:

a	a	a	1	0	0
a	a	a	0	1	0
a	a	a	0	0	1

We fiddle this into the form

1	0	0	d	e	f
0	1	0	d	e	f
0	0	1	d	e	f

and conclude that the matrix on the right IS the inverse A^{-1}.

The inverse does not exist if we fail in our attempted fiddlings! Now we can enjoy the example worked out in our problem here. Since he goes all the way and reduces the matrix A to the diagonal identity, we would call this method **Matrix Inversion by Gauss-Jordan Elimination**.

26.18. So here we redo the solution of our original Ax = c problem by using the A^{-1} we found in the previous problem. He comments that it is worth getting A^{-1} if you have to work with lots of c's.

MATRIX INVERSION BY THE EXCHANGE METHOD (this is really the Simplex method!!)

This is a strange little method. You take your starting system as shown on page 345. Then "exchange" x_3 and c_3 to get a new but equivalent linear system. The rules for doing such a swap are easily stated, and it is all shown in full detail. If you then just do this exchange three times, your array labels are all switched, and you conclude that the resulting matrix is A^{-1} . If certain elements are 0, you will be blocked from doing this since you need inverses of elements at your "pivot points" (3 of them in this case). That is how you will know that you don't have an invertible A. In problem 26.1 we do this and end up with our now-familiar A^{-1} matrix!

MATRIX INVERSION , AN ITERATIVE METHOD

The formula here is easily derived. The idea is to get an approximate solution (maybe with your 3-digit computer which "could be a slide rule") and call that B. Then the series shown improves B. In the example shown, we use the usual Gauss-Jordan elimination (with only 3 digit resolution) to get our rough solution B. Then we compute R, and then the corrections RB, R(RB) and so on. The result is then good to 7 places only going through this R^2 term. Very nice I think!

EVALUATION OF DETERMINANTS

This is pretty amazing. Brute force determinant calculation requires n! operations which is exponential in n for large n by Stirling, so you simply cannot do it this way for n = 100. Even for n=10, n! = 3 million which is a lot of operations. For n = 20 you have 2.4 e18 operations!!

Now, if you go ahead and do the Gauss-Jordan elimination, your only changes in the magnitude of the determinant are the pivot divisions, and you may get a change in sign depending on how many row swaps you did. So the answer is basically just the produce of the pivots. This means that you can compute the determinant in this way in order(n^3) operations. For n = 20, that is 8000 operations, no problem.

EIGENVALUE PROBLEMS, THE CHARACTERISTIC POLYNOMIAL

26.27 We are back to Ax = c as usual, but the eigenvalue problem is Ax = λx or (A-λI)x = Bx = 0. Eigenvectors are merely scaled by A. Should think of solutions as being dependent on λ. If λ is not an eigenvalue, then Bx=0 only has x= 0 as a solution.

26.28 Here is a way to obtain the characteristic equation. Write the system Bx=0 where at first all λ occurrences are on the diagonal in the form (a_{ii} - λ). At this point it is claimed that you can fiddle the equations to cause all the λ dependence to be in the rightmost column. This is always done by multiplying one equation by λ (just a constant), then doing a subtraction. After doing this, you do the usual Gaussian elimination to clear the lower left triangle. During this process, since you are only adding multiples of rows, the λ stuff stays in the rightmost column. Then when you look at your last equation $Kx_n = 0$, you will find that K = K(λ) is the characteristic polynomial. If λ is NOT a root of it, then $x_n = 0$, and as you work backwards, doing the backward substitution, you get that all of x = 0, and that means Bx=0 has only trivial solutions.

A theorem is quoted in passing: if A is real symmetric, then all λ roots are real (though perhaps not distinct) and n independent eigenvectors exist.

26.29 Another example is given, this one with 4 equations. $K(\lambda)$ is not solved in this case, however.

26.30 In this example there are three λ, but two are the same. We have a "loss of pivot" situation here. I did not follow the details.

THE POWER METHOD

26.31 This **power method** is similar to something we saw earlier. Imagine that there is a "dominant root", which means one single root has larger mag than all the rest. This I guess means the largest mag root is neither degenerate nor is a CC pair. In this case, just order the roots and make λ_1 be the largest one. Let the V_i be the eigenvectors of the roots of the char equation. Let V = a general lincom of the V_i with coeffs a_i. [We have to assume that $a_1 \neq 0$ in our choice of V.] If you compute a power like A^pV, then as p gets larger and larger, you reach the limit $A^p\mathbf{V} \rightarrow \lambda_1^p\, a_1\, \mathbf{V_1}$. This is because all the lower terms contain ratios like $(\lambda_3/\lambda_1)^p$. Suppose you are out in the large p region. then consider the above as n scalar component equations which I could write as $[A^pV]_k = [\lambda_1^p\, a_1\, \mathbf{V_1}]_k = \lambda_1^p\, a_1\, [\mathbf{V_1}]_k$. Then do the next term which is $[A^{p+1}V]_k = \lambda_1^{p+1}\, a_1\, [\mathbf{V_1}]_k$. Then take the ratio of components $[A^{p+1}V]_k / [A^pV]_k = \lambda_1$ and this gives you n different estimates of λ_1. If the estimates are pretty close and things don't change with the next larger p, say, then you have arrived at λ_1 and then of course you know that $A^p\mathbf{V} \approx \lambda_1^p\, a_1\, \mathbf{V_1}$ is your unnormalized eigenvector.

26.32 An **example of the above** with a n=3 system. Start with V = (1,1,1). Compute A^7V and the next and look at the three ratios, and in this case they are all about 3.414 so we may conclude that the largest λ_1 is that number. If we rescale the A^8 resulting vector, we get (1, -1.414,1) which is very close to the exact answer. So here again is a "numerical method" of finding the largest eigenvalue and its eigenvector.

26.33 If you think you are close to a correct $\{\lambda, \mathbf{x}\}$ for some A, then the ratio x^TAx / x^Tx (known as the **Rayleigh Quotient**) might let you improve your value of λ. Clearly if $\{\lambda, \mathbf{x}\}$ is exact, this ratio is λ. So the input here is your estimated eigenvector x, and the output is an improved λ. Theory is not provided to explain why or when this quotient will improve your λ. You can see, however, that is a sort of global thing, and global things ought to be more accurate than local things.

Notice that taking the Rayleigh quotient is an alternative to taking the ratio of adjacent powers. In other words, above we had $A^p\mathbf{V} \rightarrow \lambda_1^p\, a_1\, \mathbf{V_1}$. The thing on the right is our approximate eigenvector, so call it **x**. Then use Rayleigh to get $\lambda_1 = x^TAx / x^Tx$. It is of course similar to taking one more power.

26.34. **The "other extreme eigenvalue" method.** Again, this method is an idea without good theory support. You start with $Ax = \lambda x$. Now subtract qx from both sides, where q is some arbitrary real number, then you have $(A-q)x = (\lambda-q)x$ which we can write as $Bx = \lambda' x$. This is a different eigenvalue problem. The claim is that if you pick "the right" general value for q, then solve $Bx = \lambda' x$ for its dominant λ', then you might find that $\lambda = \lambda' + q$ is the *smallest* eigenvalue of the original problem $Ax = \lambda x$. [other extreme means smallest here]

We consider the system of two problems back which had dominant $\lambda = 3.414$. Exact solution is shown bottom of page 348. We use a value of q in the ballpark of the largest λ value, q = 4. We then write

out B = A - 4I. We start again with V = (1,1,1) and we end up with λ' = -3.414 being the dominant root of the B problem. Then $\lambda = \lambda' + q$ = -3.414 + 4 = .586, and this is indeed the smallest root of the original problem. Again, zero theoretical support is provided, but at least we become aware of the idea.

26.35 If you have $Ax = \lambda x$, then trivial to show that $A^{-1}x = \lambda^{-1}x$. Then you can use the power method with A^{-1} to **find the smallest eigenvalue** of the A problem. Our standard example is done in this manner, and we regenerate the same smallest λ that we found in the previous problem.

26.36 The question here: **how could you find the second largest eigenvalue**. One plan is to first find the largest λ_1 and its eigenvector V_1. Then restart the problem with a V which is perp to V_1 so that $a_1 = 0$. Then the logic of problem 26.31 tells us that our limit should be λ_2 and V_2 . Author shows that this works fine if you start with V = (-1,0,1) which is perp to V_1 and you find $\lambda_2 = 2$. This is lucky guess because this just happens to be V_2 . If you start with some other perp thing like (0,1,1.4142) which is also perp to V_1, the power approach starts taking you toward $\lambda_2 = 2$. We get to 1.996, but then further powers start redirecting us to the λ_1 solution due to rounding error!! (almost entertaining says Scheid).

26.37 Systematic way to find all the eigenvalues and eigenvectors. **The Reduction Method**. You first find the maximal λ_1 and its V_1 normalized so first component =1 (do same with V_2). Construct $B = V_1\, r$, by which we mean $B_{ik} = (V_1)_i\, r_k$, a matrix, where r is the first row of A. Follow the algebra and you end up with a new problem which is $(A-B)(V_2 - V_1) = (\lambda_2-\lambda_1)(V_2 - V_1)$ which reduces to $A_2\, y = (\lambda_2-\lambda_1)y$ where now A_2 and y are reduced in dimension by 1. You could do this because the first row of (A-B) is all zeros, and the first component of $(V_2 - V_1)$ is zero. This allows you to cross out the first row AND column of (A-B) and what is left is called A_2. You then solve the problem $A_2\, y = \lambda' y$ in this n-1 dimension space and you find the dominant eigenvalue of THAT problem λ', and then you know that $\lambda_2 = \lambda_1 + \lambda'$. The idea is that you completely subtract out the dominant eigenvalue and eigenvector from the problem and you then have a new problem where the largest eigenvalue is now λ_2.

You can iterate this method and ultimately find all the eigenvalues. Of course we have the usual issues of complex pairs and multiplicity to deal with, and those details are unmentioned here. The general idea looks very good to me.

JACOBI's METHOD

26.39. This method is bizarre. First we state the usual theorem that if A is real symmetric, you can bring it to diagonal form with some real orthogonal matrix O. The diagonal elements are the eigenvalues, and the columns of O are the eigenvectors.

Now the idea is to try to build up the matrix O a little bit at a time. The example of the next problem is where to look right now. The matrix there shown as O_1 is a little 2x2 rotation designed to diagonalize the upper left 2x2 piece of A. Obviously you can find an angle ϕ which does this, and the claim is that $\tan 2\phi$ = some function of the a's. This is the trivial part.

Now having done that, call the resulting partially diagonalized matrix A_1 and this is shown top right on page 353. Now take a different 2x2 rotation (this one xz instead of xy) and clear out two other elements of A_1 . This gives A_2 as shown. Yes, it is true that the previously cleared elements have now been dirtied with small numbers here 0.3. Accept this fact.

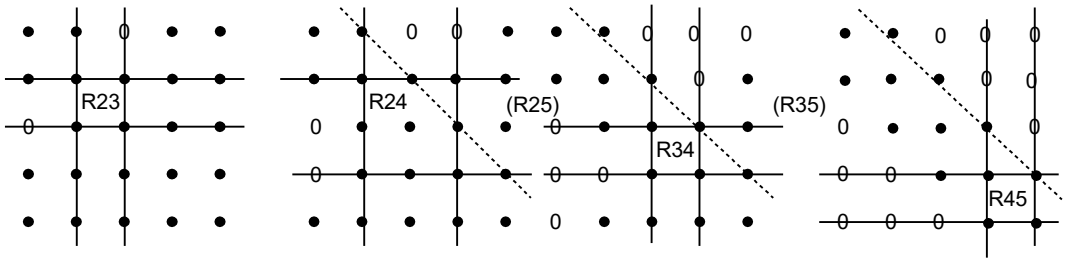
Now next you probably should apply a yz rotation to clear out the numbers -.6 which you see in A_2. that makes A_3. Then you just keep repeating the application of your three rotations, of course tuning each one to clear to off diagonal elements.

The claim is this: as you do this, the A_k matrices so formed will converge to a diagonal matrix. Ie, your sequence of operations O_i converges to O. No proof is provided, but is seems reasonable. Imagine trying to rotate a 3D object into some position. You can surely do this by a sequence of small adjusting rotations about the various axes.

26.40. Here we do our standard 3x3 example using Jacobi's method. Scheid does not say exactly what 2x2 rotation sequence he used, but with some sequence he ends up with A_9 as shown. You see it is perfect to 5 decimal places, and so are the eigenvalues.

GIVEN'S METHOD

26.41. There are many things going on at once in this section. First, think of this as a milder version of Jacobi above. In the previous Jacobi, latter rotations messed up the 0's of earlier ones. If we are willing to diagonalize only to a tridiagonal form, then it is possible to do a finite set of 2x2 rotations to get this tridiagonal form. Here is the general way this would be done for a 5x5 matrix:



A rotation (similarity) affects 4 lines. An element on only one line is "affected" by two elements in the original matrix, the one at his position, and the partner on the other line. Mixture is a cos ± b sin as usual. We can select any element on a single line for annihilation, and because original matrix is symmetric, so is the rotated result, so the transpose element is also cleared. If matrix has 0 0 at the pair, then the 0 0 pair will stay. That is why in the third rotation shown above, the top pair of zeros remains as a new one is added. Missing rotations in the above sequence are shown in parens. The result is tridiagonal as claimed!

The point here so far is this: **Using a finite sequence of 2x2 rotations, you can bring any matrix into tri-diagonal form**. Just do the procedure shown above. Since the rotations are orthogonal, if the original matrix is real symmetric, you have not altered the λ spectrum of the matrix. Recall that tridiagonal plays a big role in Jim's papers and we did associate the name Jacobi with these matrices somehow.

Call the tri-diagonalized matrix B. The next step is this: Let $f_i(\lambda)$ be the sequence of determinants of the matrix $B-\lambda I$, working down the diagonal. The tridiagonal form makes these determinants relatively easy to compute, and in fact you can write a recursion for them as shown on page 353. [Note the probably traditional use of α and β. I forgot to say: it must be that applying O to a real symmetric matrix preserves the idea of real symmetric, so B is real symmetric, so the off-diagonals are equal.] The last element in our little recursion is $f_i(\lambda)$ and since this is the entire determinant of $B-\lambda$, it is the same of that of $A-\lambda$, and we know that this gives you the characteristic polynomial. Scheid has not mentioned this before, that $|A-\lambda| = 0$ is the char equation. The reason is that for λ which solve this, you can have non-trivial solutions of $(A-\lambda)x = 0$. So one thing I guess we can say is that we now have a recursion method of computing the determinant using the tridiagonal form.

Suppose we then go off and find the eigenvalues. The sequence of $f_i(\lambda)$ determinants happens to be a Sturm sequence, so it can tell you generally where the roots are located, then you can use Newton to zero in on them.

Next, we want the eigenvectors. The tridiagonal form is awfully close to the Gauss elimination form already, but you do have that lower off-diagonal nonzero. If you write $BU = \lambda U$ for some selected eigenvalue λ, because B is tridiagonal, each component of this equation only has (at most) three terms on the LHS and 1 term on the RHS. The first and last equations have only 2 terms on the LHS. You then end up with n equations in n unknowns. But the equations are so simple, that you just set one component to 1, and work your way up the rest of the equations and just read off the results.

So to summarize: (1) it is possible to convert A to tridiagonal B with a finite sequence of 2x2 rotations. (2) use the recursion to compute det B and you then have the char poly; (3) Solve this any way you want for the λ. (4) for each λ, read off the eigenvectors after deleting one equation. So this seems to be a complete method for solving an eigenvalue problem $Ax = \lambda x$. The previous Jacobi method was another way of doing the same thing, but it required a limit, whereas here we need take no limits.

26.42. An example of Given's Method is done with the nasty 3x3 Hilbert matrix. We get the tridiagonal B at once. We get the char poly. We use Sturm to locate the regions of the roots. We use Newton to get the roots. One root is $\lambda_1 = .002217$. [This smallness is probably why Hilbert is nasty.] [Note there is a factor 1/13 in this problem that is scaled out, that is why he has λ and ℓ being different.] Finally, for this λ_1 he writes the n=3 equations, deletes the last one, assumes $u_2 = 1$, and reads off the eigenvector. The key fact is that the little equations at the top and bottom of the matrix only have two u_i in them, such as u_1 and u_2 at the top.

Theorem: The eigenvalues of an upper or lower triangular matrix are the diagonal elements.

Proof: Consider $(A-\lambda)x = 0$. The char poly is $\det(A-\lambda) = 0$. But for a triangular matrix, the det is just the product of the diagonal elements from the usual det formula. Thus the char poly is $\Pi\ (a_{ii} - \lambda) = 0$. The char poly is then in factored form and you just read off that $\lambda_i = a_{ii}$, the diagonal elements of A.

Comment: Recall that Gaussian elimination gets you to an upper triangular matrix. Suppose we do Gauss elimination on $(A-\lambda)x = 0 = Bx$. $B = A-\lambda$. I don't think the various row operations are representable by orthog transformations. If they were, then Gauss elim would get to triangular form and the eigenvalues of your problem would be the diagonal elements of B. I don't think this works or Scheid would have mentioned it. It seems that shuffling rows is not going to be a simple rotation. (!!)

COMPLEX SYSTEMS

26.43 Here he shows that if you have an n x n problem Cx = d where C and d are complex, you can do two separate things:

(1) you can reduce it to a 2n x 2n problem where everything is real.

(2) you can reduce it two separate n x n problems.

We can then apply all our previous methods to these smaller all-real problems.

26.44. **Inverse of Complex**. If C is a complex matrix, how can we find C^{-1} ? The answer is to write C = A + iB and just brute force compute the results as shown.

26.45. This is **Jacobi for Complex**. Recall the parallel theorems that real-symmetric (Hermitian) matrices can be diagonalized by orthogonal (unitary) matrix transformations with preservation of the eigenvalue spectrum. In the real case, we worked with 2x2 rotation matrices to clear out off-diagonal elements in a 2x2 region of our matrix. Here we use 2x2 "unitary rotation" matrices to accomplish the same thing. Instead of a single angle in the former case, we write our little U 2x2 with two angles. Remember that unitary means = star transpose inverse. He shows how you can solve for the angles to get a little matrix U that diagonalizes a 2x2 piece. So apart from this, it is all the same. Apply an infinite sequence of U rotations to bring your complex Hermitian matrix A to diagonal form.

26.46. **The Halfway Jacobi Method**. Use the complex Jacobi above to get A into upper-triangular form T. That is, this is the infinite sequence thing as in regular Jacobi. [But here we are not working to clear the other triangle, so I call it halfway.] Once you have cleared one side, you know that eigenvalues are the diagonal elements. Then it is very easy to construct the eigenvectors of the triangular T as shown, and then you apply U on these to get the eigenvectors of the original A.

So I guess I am now wondering why this method was not mentioned in the real world? It's sort of a half-Jacobi operation. I guess since we have it here in the full complex sense, no reason to repeat it earlier.

****

Chapter 27: Linear Programming

THE SIMPLEX METHOD

27.1,2 This example is in E^2 so we have n = 2 and m = 3 because there are three extra constraints in addition to the n constraints that $x_1 \geq 0$. Note that we can write all 3 of these extra constraints as $Ax \leq b$ where dim(x) = 2 but dim(b) = 3. Note that A is NOT a square matrix in general, so we are in a "different world" here from the usual matrix linear algebra.

Within the boundaries of the n+m = 2+3 = 5 constraints, we are supposed to maximize some linear function H of the x_i . In these two problems we use two different H's. In each case we notice that the solution seems to occur at a "vertex" of the constraint volume. This seems very reasonable to me. A vertex is defined by your being on 2 boundaries at once, here n = 2.

Hyperplane Normal Theorem: The equation $\mathbf{a} \bullet \mathbf{r} = 0$ defines a hyperplane in E^n that passes through the origin. The equation itself tells you that can be regarded as the **normal** to this plane, because no matter what r you pick on the plane, $\mathbf{n} \bullet \mathbf{r} = 0$. Suppose you go to a different coordinate system that is just translated along one or more axes by some amount. You will then have $\mathbf{a} \bullet \mathbf{r} = K$. This tells you that in general, the equation $\mathbf{a} \bullet \mathbf{r} = K$ defines a hyperplane with normal a. Translating a hyperplane does not change its normal, so you can translate it so it passes through the origin to see that $\mathbf{n} = \mathbf{a}$.

For E^2 we usually think of y = mx+b but write this as mx-y = b. Translate to pass through the origin and you have mx-y = 0. The normal to this line must then be (m,-1).

27.3 In this E^3 example, we have n = 3 and there are m=2 extra constraints so n+m = 3+2 = 5. Variables are called y_i here just to confuse us. We now want to minimize a lincom of the three y's which is a plane with a certain normal. The solution is at an intersection of 3 boundaries. In general, solutions are always

at an intersection of n boundaries, because this is a single point in E^n. Note that in this example, it happens that the bounding volume is open, whereas in the previous n=2 examples it was closed.

27.4 From the examples, we are made comfortable with these claims (which go unproved)

(1) the constraint volume (set of feasible points) is always convex. Any two points inside are connected by a line that is also inside. This is basically because the walls are all flat.

(2) The "corners" of the volume are points in E^n where n boundaries meet. These are called the extreme feasible points.

(3) A solution always occurs at one of these corners.

27.5 A horrible presentation. This section is supposed to "explain" the simplex method, but it completely fails. I had to look elsewhere. Still, I will try to follow his steps in these notes as best I can.

We replace the m inequalities $Ax \leq b$ with m equalities $Bx = b$ where B is the augmented matrix shown: $B = [\ A_{mxn} \mid I_{mxm}\ ]$, and where we have added m new variables x_{n+1} through x_{n+m}. We could have called these extra variables something else, but by calling them x's, we get the nice $Bx = b$ as our one unified statement. Each extra variable x_{n+i} provides a measure of how far we are from the boundary b_i. Of course the initial x_n variables tell us how far we are from the axis boundaries. Remember that the axis boundaries are ALWAYS considered part of the volume boundary, so we always have $x_i \geq 0$.

Therefore, all n+m components of the augmented vector x are measures of distance to a boundary. Things are on an equal footing. The extra m components we added are called **slack variables**.

Clearly, an **extreme feasible point** (corner) can only occur where n of the n+m x components vanish. How many such corners are there: $\binom{n+m}{n}$. The basic idea is going to be this: compute your H at each of these corners and you know your solution is the corner which has the max or min H.

Starting at the mark shown, I am no longer able to follow Scheid's discussion, so I will now have to replace it with my own words. [I went off and read Glicksman's book at this point.]

OK, here is what Scheid is doing. Equation (0) is our set of constraint equations and we can interpret the columns of the table as vectors in an m-dimensional vector space. Since each column has m components, this is an E^m space. At this point Scheid does something I would not have done. Instead of taking as his initial feasible solution the point $x_i = 0$ for i = 1 to n (which is what other books do) , he takes the point at the other end $x_i = 0$ for i = n+1 to n+m. This choice makes everything unpleasant, but it is OK to do it.

Now equation (1) is a rewrite of (0) but at the starting feasible vertex point, so the last n terms are missing. Notice that this might include both unit vector columns and other columns in the table, because we don't know whether n<m or the other way around. Fine.

Equation (2) is a statement of the value of H(x) at our starting vertex. Think of that as vertex 1, so this value is H_1 (he never really explains that subscript). Normally H(x) has all n+m terms, but as before, the last n terms are missing. When other people do this description, they call the leftmost n columns the non-basic variables and the rightmost columns the basic ones. When you start out in the standard way, you set the non-basic variables all to 0 (ones on the left), and you have nice unit vectors for the basic variables (on the right), The variables on the left are the "physical" variables of the problem, and you always start with H being a function of them, and there are n of them. But here, we H being a function of m variables x_1 through x_m some of which might be basic and some non-basic, so this is quite confusing. When doing a problem, you don't have H in this form. Still, it is OK to write it, the equation (2) is at least true. Note that the coefficients c_i are defined for all i = 1 to n+m, as shown at the top of the page. When you start in the

normal fashion, you would have H = 0 or maybe H = the constant term in your initial H equation, and that would be your H_1.

Moving right along, we come to equation (3) which just says you can write all column vectors in the table as lincoms of the set you select as your basis vectors. Again, instead of selecting the last m columns as the obvious starting unit vectors, Scheid takes the first m columns which are messy and it is not obvious to me that they are even linearly independent. He "assumes" they are. So we have a messy set of basis vectors. Again, OK to do it this way. Now comes another confusion. The coefficients in the linear combinations just mentioned might have been called K_{ij} or something like that. But he calls them v_{ij}. This is very confusing because you probably would like to call the component of one of the column vectors by the same name. Perhaps $(v_i)_j = v_{ij}$. But these numbers are unrelated to the v_{ij} he is using. Very ugly! For example, the $(v_i)_j$ are a_{ij} for the first n columns, but the v_{ij} are δ_{ij} for the first m of these columns.

Now we come to the horrible equation (4). For the moment, just take this to be the definition of the weird quantity h_j. Since c_j is defined for all i, no problem with h_j having same range.

At this point, let me pause to point out that Scheid does not comment that you can consider equations (1) and (2) to be part of the same matrix equation with a new variable M, the way Glicksman does. This single addition of G's makes everything much clearer.

Now we come to equation (5). At our starting position, we have all x_{n+1} through $x_{n+m} = 0$ in Scheid. We are on some vertex of the convex volume in E^m (but he does not say that). Note that this E^m is the space where he has set *his* starting variables at the origin. This is not the same E^m physical space that one usually starts with, but that is OK. We then pick one of these zero variables and call if x_k . We ask what happens if we change from $x_k = 0$ to $x_k = p > 0$. What does this mean? If we take one of these zero'd variables x_k and move out onto the positive x_k axis, we are moving away from our starting vertex in E^m along *one* of the axes that attach to that vertex (axis k). This is because all the other "axis" variables are still 0. As we move away from a vertex, we expect all the variables other than x_k to change. I like to think of them as decreasing because that is what happens in simple 2D problems when you start.

This is what equation (5) is saying. First, you can strictly derive equation (5) by starting with (1) and then you add and subtract p $\mathbf{v_k}$. Since v_k is not one of your current basis vectors, you expand it in terms of the current basis and you then distribute the terms to the appropriate terms in (1). This gives you directly equation (5) and that is all there is to it. I did this in a separate document somewhere. But the interpretation of (5) is that if you poke out amount p in the x_k direction, moving away from your current vertex, then you are going to sort of reduce all the other variables. You might define $x'_i = x_i - pv_{ik}$ and then you say that your new coordinates are x_i' for i=1 to m and x_k.

We come now to equation (6). You can strictly derive this as well from earlier equations. If you insert into (6) the definition of h_j given in (4), all those v_{ij} terms cancel out and you are left with (2). Notice that h_j has that extra c_j piece on the end which offsets the pc_k you see sitting in (6).

Now the interpretation of (6) is the harder part. You have transformed your coordinates by sliding out in the k direction from your vertex, so H has changed. The LHS of this equation is H(x), so this equation says that $H(x) = H_1 - ph_k$. So, if you are trying to minimize H [we are], then if this strange quantity h_k is positive, then you are improving the situation by moving along this "edge" away from your vertex.

Obviously, you want to go as far as you can go with p since larger p helps make H(x) smaller. You go as far as you can until the first "other variable" hits zero. That is to say, you arrive at the vertex which is at the end of the edge you are moving on. Remember that the variables of our E^m space are x_{n+1} through x_{n+m} and, but all those boundaries "out there" are represented by the other variables. By choosing p as shown in (7), you get to the next vertex. You can see all these factors $x'_i = x_i - pv_{ik}$ sitting in (6) or (5), and we want to increase p until the first of these becomes zero. Again, this is all strictly true in Scheid, but we

have all these v_{ij} floating around which we don't really have much knowledge of. The rules of course says pick the minimum of the ratio shown but only for positive v_{ij} .

So, if we move by this value of p, we arrive at what we might call $H_2 = H_1 - ph_k$, and we are now at vertex 2.

Now comes another observation. Since p clears out one of those factors, call it factor ℓ, in equation (5) which is our matrix or table equation, we no longer see $\mathbf{v}_\ell$ but we do see the new $\mathbf{v}_k$. So **b** is now being written in a new basis where we have swapped $\ell \leftrightarrow k$. That is true, but Scheid gives us no fell for this at all. We don't see a unit vector moving from one column to another in his table. Because his basis is weird, we can't really see what is going to happen in the table as we make this motion from one vertex to another.

He then goes on to talk about the basis vector swap on page 365 using equations which for me, relate the v_{ij} which I don't care about to the new v_{ij}' which I don't care about. He then ends up with a new table that as far as I can tell, has nothing to do with anything. It is not the simplex table. It is not something that is a matrix and this is a shorthand notation.

Scheid refers to what he is describing as the simplex method, but he does not show the table, he does not show the M equation, it just could not be more poorly done. Scheid is usually so good, I wonder why this section is so bad. He makes no mention of Gauss-Jordan. No geometric comments on what is happening, it is all just cold meaningless equations to the reader. We can derive them all, but they all we have to say is "so what?" .

27.7 (Simplex solution of 27.1) Now we come to our first simplex problem, and Scheid ignores everything he has described in the previous problem and does things the normal way here. We have m = 3 conditions, so we get 3 slack variables. The x1 and x2 are the starting physical variables, so our mapping is $E^2 \rightarrow E^3$ for our matrix equation if we think of it that way. We are to minimize $H = x_1 - x_2$ (as usual, presented in the physical E^2 coordinates). On page 366 he draws his first simplex table. Stub is on the left. Instead of labeling the table with x_i coordinates, he persists in his basis vector notation. So fine, $\mathbf{v}_1$ is the basis vector for x_1. The last row is the "objective" but Scheid does not explain how it gets into the table! He does not associate this table with a matrix equation. There is no M variable for the last row. He refers to the numbers in the bottom row as the h_j but I don't see why that is the case. I see now through the eyes of other teachers of simplex.

Now we do the algorithm. We pick the most positive item in the bottom row to get our pivot column. (Remember that Glicksman picks the most negative because G is maximizing, S is minimizing.) That determines the pivot row. We circle the winner of our smallest ratio test, and that is the pivot element in the table. We then do Gauss-Jordan and slam this column into a unit vector, and we have now sullied the former unit vector for x_3 . Since all bottom row entries are negative, we are done and the answer is that the objective is minimized at the value -1.

Now this problem was previously treated as 27.1. There we see the convex region in the physical space E^2. We start at the origin as usual and move up to the next vertex, and it happens to be the winner based on the sliding line method done earlier. The answer is +1 here but he was doing x_2-x_1 max, and in the simplex version he is doing x_1-x_2 min.

27.8 (Simplex solution to 27.2) This is the same problem but with a different objective, we have $H = -2x_1 - x_2$ in the physical E^2 space. Same convex region as in problem 27.1, different line. Here we do two pivots to get the answer which is min = -7, and in problem 27.2 we found +7 by maxing. Ie, 27.8 is same as 27.2 apart from this.

27.9 (Simplex solution to 27.3) Now we want to do 27.3 by simplex, but our usual starting point origin is excluded because *one* of the 2 conditions is a ≥ one. Scheid introduces (in addition to the two slack variables) *one* "**artificial variable**" called y_6 and a "very large parameter W". This is the Big-M method described better in another document. But again Scheid flops. He fails to explain where the last row came from in his 3rd table on page 367. Nor does he explain why the W's disappear in the next table, but he does show a reasonable pivot selection, and one pivot operation finishes the problem giving min = 1.

27.10 (Simplex solution to problem that will turn out to be dual to 27.3) One more. This problem has not been done before. One artificial variable is needed, and then two pivots. We end up with min = 7.

THE DUALITY THEOREM

27.11 This I think was proven first by Lemke in 1954 and it generalized the 1928 Minimax Theorem of von Neumann. You write down problem A as shown. When you write problem B, you first write everything in transpose form, vectors on the left. Then you swap b and c. In problem A, b is the vector of boundary limits, and c is the vector of coefficients of the objective function H. So in problem B, these roles are reversed. In A you minimize, while in B you maximize.

When you have dual problems, the optimized answer for H are equal! For example, in our problems 27.1 and its dual problem 27.3, the optimized answer is 1 in both cases. There is also a way to relate the solution vector x to the solution vector y.

This does not really seem like rocket science, but no proof is given here, nor in Glicksman, so the proof may no be as trivial as I would guess it is.

27.12 Author shows that problems 27.1(27.7 simplex) and 27.3 (27.9 simplex) are dual. Both problems have solution = 1.

27.13 Author shows that problems 27.2(27.8 simplex) and (27.10 simplex) are dual. Both problems have solution = 7.

SOLUTION OF TWO PERSON GAMES

27.14 The operation of a "matrix game" is better explained in Glicksman. Both players are smart and can adjust their strategies. Here, player C (for column) takes strategy (p_1, p_2, p_3), meaning these are the probabilities with which he is going to select columns 1,2 or 3 of matrix A in a statistical ensemble of game events. The entries in the matrix A show the "payoff" that **R pays to C**, let us say, if that element is hit.

Player C can compute the size of his winnings he will get long term assuming player R does one of his three "pure strategies" (ie, if R plays only one row). Player C then picks the *lowest* of these three numbers P1, P2 and P3 and calls that P, and that is the lowest his winnings can possibly be (this would be average winning per play over a long time). Whatever strategy R picks, winnings will be this amount P or better. So knowing this, player C then seeks to adjust his strategy (p_1, p_2, p_3) to maximize his win worst case P. This is a simple linear programming problem, call it Problem C.

Meanwhile, player R does the same thing from his point of view. His strategy is (q_1, q_2, q_3), and based on pure plays by C, his average per-play loss will be the *largest* of Q1, Q2 and Q3. So he sets Q to the largest of these, and he knows his loss will in practice be this amount or less. He then wonders how to adjust his strategy to minimize his loss Q. He can do a simplex minimization problem, Problem R.

Of course the roles of win and lose depend on the signs of the P and Q so computed.

I suspect that because of the fact that the two players are using transposed sides of the same matrix, that is what causes these two problems to be **duals to each other**. The two independently arrived at strategies give the same results, so P = Q. If this is 0, it is a "fair game". If P = Q > 0, then player C is going to win on the average and this fact is built into those matrix numbers. It is not a fair fame.

27.15. Now we put actual numbers into the matrix and, for solution, we use the player's side that does not need artificial variables. The result is optimal payoff = 5/6. C's optimal strategy is $(p_1, p_2, p_3) = (3/6, 1/6, 2/6)$ while R's optimal strategy is $(q_1, q_2, q_3) = (1/6, 2/6, 3/6)$. Scheid notes that you could make the game fair by charging the winning side C 5/6's of a dollar for each play of the game.

27.16. Here we have a different matrix A. Since the central element is max of its row and min of its column, and this tells us that each player will end up with a pure strategy, and something is a **saddle** point (this was drawn in the G book and appears on his cover page as well). The optimal payoff here is -1/2 which means that R actually wins in this case, and the pure strategies are both (0,1,0).

I guess we can conclude that any two dual problems can be mapped to one two-person game with the same matrix in the case that **b** = **c** = (1,1,1...). The strategies are then the **x** and **y**.

****

Chapter 28: Overdetermined Systems

There are several things you need to review from earlier chapters for this chapter.

28.1 First, some review. In Chap 21 on least squares fitting, we did DS=0 on the least squares sum to find the normal equations. The metric here something in E^N. We can *interpret* the normal equations as $(\mathbf{y}\text{-}\mathbf{p},\mathbf{e_k}) = 0$ where **y** is a point in E^N and **p** is its projection in subspace S which has m dimensions. This is really m equations for the components of **p**, once you are given **y**, and these are the normal equations. In Chap 21, the scalar product used is a discrete L_2 sum. The solution **p** will be that **p** which is closest to **y**. In Chap 21, we had **y** = arbitrary discrete function, **p** = polynomial of degree m. We later redid both of these ideas for continuous space x. S was the subspace of polynomials of degree m.

In the current chapter, we start with A**x** = **b** where **x** is in Euclidean E^n. As **x** takes values in E^n , normally in a non-singular square matrix we can have **b** be anywhere in E^n as well. So normally, the range of A**x** is all of E^n. However, when A is non-square with too many rows (overdetermined problem), then you cannot hit all of E^n with your **b**. The range of A**x** is a subspace in E^n of dimension m = cols and m < n. Call this subspace S as earlier. The thing **b** plays the role of **y**, somewhere in E^n. The thing A**x** plays the role of **p**, lies on the subspace. We are trying to find **x** so that A**x** is *as close as possible* to **b**, where here distance is based on the usual E^3 dot product. The normal equations for finding A**x** as close as possible to **b** are these: $(\mathbf{y}\text{-}\mathbf{p},\mathbf{e}_k) = 0$ which here says $(\mathbf{b} - A\mathbf{x},\mathbf{e}_k) = 0$ which is m equations. The only question is: what are the $\mathbf{e}_k$ in this problem? They are a basis in S. If you apply matrix A to unit vectors in E^m, you get the columns of matrix A as being a viable basis for the space S of m dimensions inside E^n. Call these m column vectors the $\mathbf{a_k}$. Then we have $(\mathbf{b} - A\mathbf{x},\mathbf{a_k}) = 0$. Now we do two things. First, note that $[\mathbf{a_k}]_j = a_{jk}$ because $\mathbf{a_k}$ is the vector for column k, and the column index is the second index on a_{jk} . Second, we know that we can write $[A\mathbf{x}]_\mathbf{j} = \Sigma\, a_{ji}\, x_i$. Combine to get $[A\mathbf{x}]_\mathbf{j} = \Sigma\, x_i\, [\mathbf{a_i}]_j$ or $A\mathbf{x} = \Sigma\, x_i\, \mathbf{a_i}$.

This is the result I always have a little trouble with, that A**x** is a lincom of those matrix column vectors. Using this, we now have that (**b** - A**x**,$\mathbf{a_k}$) = 0 => (**b** - Σ x_i $\mathbf{a_i}$,$\mathbf{a_k}$) = 0 =>

($\mathbf{b}$, $\mathbf{a_k}$) = Σ x_i ($\mathbf{a_i}$, $\mathbf{a_k}$) => Σ_i ($\mathbf{a_i}$, $\mathbf{a_k}$) x_i = ($\mathbf{b}$, $\mathbf{a_k}$) k = 1..m, sum to m

This is m equations in m unknowns, the x_i. These are the normal equations, and you solve with Cramer's rule, say. And (**a**,**b**) = **a**•**b** in the usual E^n sense.

Now having said all this, we can define **r** = A**x** - **b** to be our difference vector that we are minimizing the square of by adjusting **x** to get A**x** to be "perpendicular" to **b**. This is called the residual. The idea is that, although we cannot solve Ax = b for an overdetermined system, we can find x that gets us as close as possible to Ax = b.

28.2 Apply the above to n=3 and m=2 example. Compute size of r.

28.3 Add three more equations, so now have n=6 and m=2. Compute r, and it is larger than before. The RMS error is $\| r \|^2/m = \rho$.

28.4 We can regard our normal equations as a non-singular square matrix problem Mx=B where M_{ik} = ($\mathbf{a_i}$, $\mathbf{a_k}$) and where B_k = ($\mathbf{b}$, $\mathbf{a_k}$). We can solve this in many ways, one is Gauss elimination with back-substitution, another is Gauss-Jordan.

28.5. In the least squares world, we are using the normal a•b scalar product to generate a norm which in turn makes a metric and we have a Banach space deal. Suppose instead you want to minimize max($| r_i |$) which is sort of an L_1 thing. The problem is this, I think. In the previous problem, we are not dealing with functions (discrete or continuous), so we are not dealing with the traditional L_2 norm an integral or sum over points on a segment or mesh. We are not doing a polynomial fit here. In fact, the scalar product was not the L_2 one, it was the Euclidean one a•b. In the second problem, what do we do? Yes, the idea that $\| r \| = \max(| r_i |)$ is a norm, I just showed it page 105 Stakgold. But there is no scalar product here! In the previous example, we had $\| r \|^2 = (r,r)$, but in our second case we are *missing* a scalar product. We have a metric space and a normed linear space, but not an inner product space, not a Hilbert space. Thus, we cannot apply the previous method! Scheid does not mention all this.

So that is why we have to solve this problem with the Simplex Method, and that is why this chapter has to come after the linear programming chapter in the book.

So here we go: our $r \equiv \| \mathbf{r} \| = \max(| r_i |)$ idea results in 2n equations, n is number of original equations. These are on bottom of page 377. You might as well divide through by r so $y_i = x_i/r$ and then the RHS of all equations is 1. You then solve using Simplex. Notice that the maximal y variable is just y = 1/r.

The solution we are describing can be called the **minmax** solution or the **Chebyshev** solution. Recall that his name is associated with minmax (the line closest to 3 points) and his polys were useful as well.

28.6 We apply simplex to problem 28.2 where we had 3 equations. We need 2n= 6 slack variables (notice that we don't need any artificial ones since ≤ constraints). He now writes the Simplex table showing only the constants column and the three physical variable columns. He does the clearing out process and gets the three unit vectors after 3 pivots. He can then read off the solution values of y_i for i = 1,2,3. He gets $y_1 = 10$ and $y_2 = 3 = y_3$. This means $x_1 = r*10$ and $x_2 = r*3$ and **r = 1/3**. Meanwhile, the objective is to maximize $y_3 = 1/r$ to minimize r. But he uses $M = -y_3$ and minimizes M, so he will be

looking for positive numbers in the objective row, not negatives as I am used to. He does the three pivots until all positives are gone. So we end up with $x_1 = 10/3$, $x_2 = 1$.

Now he makes the claim that the "three residuals" all have size 1/3 as usual in a min-max situation. To show this, I would have to compute each one. For example, one equation was $x_1 + x_2 = 4$ so bottom page 377 says $x_1 + x_2 - 4 = R_1 = 10/3 + 1 - 4$ which is 10/3 - 9/3 = 1/3, as he claims. The min-max norm always gives you the three equal size errors as in chapter 22.

Notice in the example above that we did not bother to show all the non-physical columns of the table. I guess that means we don't have to compute them?? How do we know they won't have positive numbers in the objective row that we have to worry about? Well, this is exactly the condensed method described in Glicksman. There are two rules you have to follow. Actually, I think his thing is a little different, I think Scheid has just not shown the other columns, he is not switching columns around. The Glicksman scheme always shows only the current non-basic columns, and never shows any unit vector columns. Notice in Scheid that the row labels change with each pivot as usual because they go with the 1's of the unit vectors. I guess you get faster at this after a while. Need a program to do the computation.

28.7 Now we repeat 28.3 where we have added 3 more equations. We now have 6 equations, so 12 inequalities and a larger Simplex. He gets the solution now to be **r = 3/4** which is larger than previous 1/3. In this case three residuals he claims are equal in mag, others are smaller, but he does not compute them. I guess they must be 3/4 in mag, yes, since that is r.

28.8 He shows trivially that $\rho_{rms} \leq r$ for *any* solution $\{r_i\}$, where $r = \max(|r_i|)$ is the minmax r. In particular, for the LS solution with ρ, we know that $\rho \leq r$. Now consider the $\{r_i\}$ you get from the LS solution. In terms of $\max(|r_i|)$ for these r_i, the minmax solution must be better than the LS solution, since minmax is the best of *all* solutions for $\max(|r_i|)$. Therefore, $r \leq \max(|r_i|)_{LS}$. So we get this bounding or r:

$$\rho_{LS} \leq r \leq \max(|r_i|)_{LS}$$

Therefore, suppose you first do the LS problem, which is usually easy to do. You find the two bounds for y. Now suppose you want to know r. You compute the two bounds, and if the range is small, maybe you just use the average as your and avoid doing minmax simplex.

28.9 Here the 3 numbers are shown for problem 28.2. In this case the bounds are (.30, .43), so if you guessed half way, your guess would be .36 for r. The actual value is .33, so not too bad. If you use the LS solution for your guess, then .3 is not too far from .33

28.10. Here the 3 numbers are shown for problem 28.3. The range here is (.57, .92) so you would guess maybe .75 from the averaging idea, which is perfect. But if you chose the LS solution .57, you would be quite a ways off. Scheid's wording is a little hazy in these last two problems, but I get the point.

****

Chapter 29: Boundary Value Problems

Here is the naming convention,

Second-order PDEs of two variables are of the form

$$a\frac{\partial^2 f(x,y)}{\partial x^2} + b\frac{\partial^2 f(x,y)}{\partial x \partial y} + c\frac{\partial^2 f(x,y)}{\partial y^2} + d\frac{\partial f(x,y)}{\partial x} + e\frac{\partial f(x,y)}{\partial y} = F(x,y). \tag{3.1}$$

and can be classified on the basis of the discriminant, $b^2 - 4ac$, into three types based upon the conic sections of the same name:

$$\begin{array}{lll} b^2 - 4ac > 0 & \text{hyperbolic} & (3.2) \\ b^2 - 4ac = 0 & \text{parabolic} & \\ b^2 - 4ac < 0 & \text{elliptic} & \end{array}$$

I wonder why the nature of solutions changes at these odd seeming boundaries? Claim is this

parabolic	one linear, others seconds:	diffusion, Schrodinger (complex numbers), solutions die out
hyperbolic	all seconds, but one is different in sign:	waves, solutions propagate
elliptical	all seconds, all signs the same:	Laplace, Poisson, no time

LINEAR ODE's

29.1. Here we look at the problem L = D^2 - p(x)D -q(x) such that Ly(x) = r(x) subject to an rather arbitrary looking BC's at x=a and x=b. So this includes lots of possibilities, including the initial problem we know how to solve from an earlier chapter where everything must be BC'd at the same point. He shows that the solution is easily found from solving three separate and simpler problems, each of which has standard "initial" BC's. Only the third solution involves r(x). All fine.

29.2 Here we take the same ODE as above, and convert it to a difference equation in the usual way, that is, we do a central version of y" and a central version of y" (which is a little unusual but OK). This turns the ODE into a DiffE with 3 offset terms in y, so it is basically a recursion formula for y. If we write out all the terms with a mesh , we get a matrix equation in the form My = b which appears to me to be in **tridiagonal** form, but the off diagonals are different. He calls this a **band matrix**. If you solve this thing by Gauss or Gauss-Jordan, there is minimal work because you only have to clear a few elements! This is very Jim-like, by the way, a payoff for marching through Scheid this far. I can see pretty much how you could do this, and the class of problems here is quite general. Any second order ODE in fact. As you make the mesh h smaller, answer bets better.

29.3 Here we examine a simple problem with constant coefficients. Solution is clearly sin(x)/sin(1), he has a typo. We convert to a difference equation with constant coefficients. We can solve this by method on page 184 where we have to solve for R and θ. This was based on the little powers method trick. It is then clear that the discrete solution approaches the continuous one, so this is an example of the numerical solution **converging**.

29.4 Here for a slightly different problem (different BCs) he writes out the tridiagonal matrix, but he does not do anything with it. He claims a sine solution, but that can only come from the other method I think.

For this problem, the vector on the RHS has only a top and bottom element from page 385 because we have a homo equation. If we did Gauss Jordan on this thing, we would get that the y_i are just some numbers, it would not be obvious that these numbers were values of a sine function.

One thing to remember: if you have a 2nd order homogeneous difference equation with non-constant coefficients, it will be tridiagonal and the RHS will have two components. Jim is always taking a recursion relation and treating it in this manner, so I am getting more tie-in to his papers now.

NON-LINEAR ODE's

29.5. The **garden hose** method name comes from the picture shown. We know how to solve a problem when we specify y and y' at x=a, but here we are aiming to hit y(b) = B. Try different values of y'(a) = M and try to zero in on your target by doing iterations. Sort of regula falsi. Note that we are not assuming linearity for the function f.

29.6. Here we apply partial derivatives relative to the parameter M. We then have the idea that we have a solution F(M) = B which is perfect, and we expand this Taylor to get $B = F(M_1) + (M_2-M_1) F'(M_1)$ and this gives a way to do Newton, where M_1 is some initial solution. But we need F'(M) which is called z here, and he shows how we can convert the DE in y into one for z. This seems bit messy because in each iteration, we have to solve the DE in z using the previously found solution for y. I could imagine a Runge Kutta variation here. He is just suggesting here a way to do **Newton on the garden hose**.

OPTIMIZATION

29.7 This states the main idea of **variational calculus** and the Euler equation, see page 200 M&M to confirm the Euler equation. Note that the two ends of y(x) are glued down by some BC's.

29.8 Do an example. $F = y^2 + y'^2$ and solution is shown.

29.9.10.11. Here he introduces an idea called **dynamic programming**. I suspect this is another bottomless pit of work, so I will skip it because I don't care about it right now. He seems to divide the integral domain into a set of equal regions and work from one end to the next, but I don't see why there is a benefit to doing this. You try to find your optimal path y(x) in a piecewise method.

THE DIFFUSION EQUATION

This is something I never really encountered. It is not cleanly in mechanics, E&M, QM, etc. Reif in the brown book at least derives the diffusion equation, but no simple problems are done there. So I have never seen a solution of an elementary diffusion problem! Well, it is also "heat flow". And maybe the wave equation in some situations.

29.12 Here he allows a linear D term in the diffusion equation. He then defines a 2D mesh (for the first time in this book that I can remember) and we have a 2D numerical problem in x and t. We get our little difference equation, and it tells us how to do the problem. Here is an interpretation: we have a wire that goes from x=0 to $x=\ell$. We start the wire off with some temperature profile at t=0, perhaps by heating it with a torch. We then put heat sinks on the two ends so that T = 0 at both ends. We then ask what happens over time. So it is the diffusion of heat in this case, and our variable is T = temperature.

Now as usual put in things for the derivatives to get a difference equation. We find that if we march to the next unit in time, at each x points we have to add a lincom of the three nearest x points at the previous time step. Now this I can understand.

29.13. Here we set a= to normalize the D^2 term, b = 0 to kill the linear D term, and c=0 to kill the D^0 term. This kills off the central term in the triple sum and we just get simple averaging of nearest neighbors as we move forward. We start with a mesh in x of 4 steps of h = 1/4 each. For time we do 1/32 unit steps. He puts a constant temperature on the wire and we watch what happens. Only the center stays hot, as you would expect. This is shown in table (a). In table (b) he ups the x resolution to h = 1/8 and time to 1/128 and things look better. We see the wire edges start to cool first, center stays hot, etc. In table (c) we go back to the h=1/4 mesh for x but set time = 1/16. It turns out that such steps are TOO BIG in time, and the solution goes unstable! The numbers are garbage.

29.14. A Taylor estimate for truncation error, but I don't follow it.

29.15 Back to our same simple diffusion problem. This time we put a sin(px) profile on the wire. The analog solution is quoted, and appears to be separable in t and x. The difference equation is written and we do "**separation of variables**" in the discrete domain, and the solution is found.

29.16. For a given h, if the mesh steps in the other direction get too large, you blow up. Here for our problem he shows that you need $\lambda < 1/2$. This means time steps are small, and you have slow going.

29.17 Our same old problem, but we use analytic separation to show that time is an expo, and x is a sine cosine thing. To meet the two x boundary conditions, the sine argument is quantized. We have a solution for each integer n. We then lincom this with some A_n coeffs to match the t=0 wire profile function F(x). So this is a **Fourier series in the x dimension**.

29.18 Another approach: leave time continuous and space discrete. Treat each space step $T_m(t)$ as a different variable in a **linear system** problem. The recursion thing is then a system of M equations say, and each equation is 1st order in time. We have done these before.

29.19. Continuing with previous: each equation in your linear system will have an expo eigenvalue solution, so just each one that way.

29.20 Another approach: Here for each time step we compute things in terms of the previous time step as we did originally. Write this M times for incrementing values of the x mesh index called m. But you then have M values of the function. Solve this MxM system at t= 0. Then move to t = 1. The idea is that you solve everything all at once at each time step, rather than do little regions as we first did. This is supposed to fix the instability problem.

29.21. In our heated wire problem, suppose you fix one end at x=0, but you let the wire stretch or shrink as time moves on, so other end is at X(t). In this particular case, the wire starts at zero length! Well, his only comment is that this type of problem appears **when boundaries move** as in melting something like metal or water on a lake. He talks a long while about how you would solve such a problem, but I skip.

Review: in this section, we had linear time and we have T_{xx} so equation was $T_t = T_{xx}$ with BC's. This was the "parabolic" case.

THE LAPLACE EQUATION (elliptical)

29.22 Now we have $T_{xx} + T_{yy} = 0$, my old friend. Numerically, on a 2D mesh you find that each point is the average of its four neighbors! You apply a BC around the entire perimeter in this equation. In E&M this is potential theory. Potential on a boundary determines it inside. If constant on boundary, it is constant inside. The max and min have to occur on the edge somewhere.

29.23 Here was have a great application of the Gauss-Seidel relaxation method. Just apply your BC's around the edge in the example here of a few points. He puts 1's on one edge and 0's elsewhere. Then just do the iteration in a fixed order repeatedly through the lattice. After a while, it stabilizes to the solution. This is the kind of thing I have long wanted to see but never saw, how you do these things numerically. This is a big reason why I am reading Scheid.

A CONVERGENCE PROOF.

29.24 He wants to show that the system of equations we used in Seidel always has a non-singular matrix. Here is an interesting argument. Suppose the max M is at some interior point. You cannot have that occur unless all four nearest neighbors also have the same value, just due to the averaging rule. You continue this argument and you will hit a boundary and that point will have to have this max M value too. But if we assume our boundary is all 0, then we must have M = 0. Basically, this shows that everything inside has to be 0. So when you think of the matrix equation Ax=b where b = boundary values are included, then b=0 implies x=0 which means A is non-singular. This argument also shows in a very simple way why the max and min have to occur on the boundary! If they occur in the middle, everything is constant on the boundary, and the whole picture is constant. Very nice.

29.25, 26. These are convergence proof which I skip.

MORE GENERAL PROBLEMS

This is a section addressing a few technical details.

29.27 What happens if your boundary is curved and misses your lattice points? Do a linear interpolation.

29.28 Too technical for me

29.29,30,31. Poisson is a driven Laplace, so your difference equation has a driving term, fine, it is still a system of equations, solve the same way. Same is true for an eigenvalue problem. Last item shows the appropriate variational thing. I think he is saying that for the function shown, Laplace is the Euler, but I don't know how to do variational in 2D yet, so let this pass.

THE WAVE EQUATION

All problems in this section: First, we write our equation in the usual difference way. This time, however, we find that the solution at A is affected by the 4 points earlier in time as shown. This seems little like the diffusion picture on page 389. However, our BC's in 29.32 are both at t=0, so "initial" in form. We get the same triangular area affected as we move in time with diffusion. This suggests to me somehow that we have a velocity limit here, we cannot be affected by things too far away. Probably this is clearer in the Green's function method where you put a delta function on the boundary and watch what happens, you will have propagator and all that good stuff. Doubtless you do that discretely as well.

Comment: I like to have physical problems that mean something, rather than completely abstract boundary conditions where I have zero feel for what should be happening.

****

Chapter 30: Monte Carlo Methods.

He first talks about random number generators on computers of base 10 and base 2. I did not know there were ever base 10 computers! Book is 1968. The problems attacked in this chapter are all different.

30.1 Gives a list of random numbers we will use in other problems.

30.2 Assume **neutrons** can do only 4 bounces before "expiring" and assume random direction and assume distance D. What percentage get through a wall of thickness 3D? This is a statistics problem where you would put in certain PDF's and get the answer. But, instead of doing that, you just shoot in 10,000 neutrons and use random numbers for your angles and see what happens. You are sort of probing the PDFs with your trials. Answer is that 28% get through, so need a thicker wall!

30.5. A strange problem. You randomly distribute N particles of mass m on a circle rim. You do lots of trials, and for each one, you note the location of the **center of gravity** and how far away it is from the center of the circle. What does this profile look like? To really solve, you put down your first one, then you put down your second particle with a uniform PDF around the rim. You then use the rules of prob theory to get your PDF through the CoM equations. BUT, instead of doing that, just do random shots from a gun and collect some statistics. Chart is shown, and for N=2, shape is arcsin(r/2).

30.6 Here is our **dog in corridors** problem again, which we NOW see is the same as the digital solution of the Laplace equation! Very interesting fact by itself. We wanted to know probability that a lost dog comes out the bottom wall if he starts at some intersection i. In Monte Carlo, you put the dog somewhere to start, then throw random choices of his four directions, not unlike the neutron problem.

To summarize, I think **Monte Carlo** is applied to problems that have a statistical nature described by some PDF situation. Instead of doing all the PDF theory and equations, you just sample the PDF with lots of samples and then bin the results and plot them. Scheid fails to comment on the name Monte Carlo.

****

Comments on Margenau and Murphy Chapter 13 ~ p 474

They mention the forward Newton interpolation formula and another one I have not seen. They mention the divided differences method for dealing with unequal grid spacing. I skipped the section on differentiation, but then read that on integration.

The **Euler-Maclaurin** (E-M) Formula is the Cadillac, and involves the Bernoulli numbers,. and it is stated in full on MM page 474. It is given there as the trapezoidal result plus corrections involving the B_i and some odd derivatives. For some reason, Scheid never really states this formula. He states it in on his p 71 in a form that relates a sum to an integral. MM show the lowest few terms of E-M on page 475.

The E-M requires derivatives of your y(x). If you just have experimental data, you don't have derivatives. In this case, you have can replace the E-M derivatives with their numerical approximations that you get from the Newton formula. This gives the **Gregory's Formula** on page 476. As usual, this formula is a function of the various differences.

The **Newton-Cotes** methods are very simple. You divide up your integration region into K intervals each of which involves n+1 points with abutment. Since the internal abutting edge points are used twice, the total points in the total interval is K*n + 1. For n = 1, each section has two points, this is the linear **trap rule**, and the total interval has K+1 points, no restriction. But if you have n = 2, you have 3 points per group, and now you are doing **Simpson's Rule**, and total is then 2*K + 1 points. If you jump up to n = 6, total interval must have 6*K+1 points, and this is the very accurate **Weddle's Rule**. On page 478 MM do a certain integral by each of the three rules.

MM then on page 479 go on to their Gauss-Legendre quadrature discussion. They claim that this method minimizes the error between the exact integral you want, but that is probably true for any choice of ortho polys. MM use the Legendre, but they use the range (0,1) instead of (-1,1) so their numbers are a little different from Scheid (they do show the connection, however). You need the locations of the zeros of your orthos, then you need to evaluate your y(x) at these magic points.

At this point, MM go on to other subjects.

Comments on Dover Hildebrand Book

Scheid is 1968, but this Dover is 1974. It is 700 pages long and has LOTS of details on everything. It uses all those fancy operators as well. This book is in my library for reference only! Even this book, however, does not treat Gaussian Quadrature in a general ortho-poly sense, but does them one at a time, see both Chapters 7 and 8. So I still don't have a book with the general approach that my notes show (but Bateman is pretty general).

My Original Introduction to These Notes (just keeping this as a reminder)

(but now chapter notes are in normal order). My notes are divided into **three** sections here. Scheid is the author of this Schaum outline book.

The **first** section involves only those chapters that lead up to the subject of Gaussian Integration. These chapters deal with functions of a continuous variable. We find the collocation polynomial p(x) that matches an arbitrary y(x) at n+1 points, and we use that polynomial to estimate things. We can find p(x) even for unevenly spaced arguments x_k . We then go on to find a formula for a p(x) that not only matches, but matches in first derivative, this is the osculating poly given by the Hermite formula. This Hermite formula is then used to develop the theory of Gaussian Quadrature. The use of ortho polys causes the extra B_i coefficients to vanish. Different weights w(x) lead to different types of Gaussian quadrature: Legendre, Laguerre, Hermite, etc.

The **second** section of notes deals with those finite differences called $\Delta^i y_0$, which is the ith finite difference at the point y_0. We are led to Newton's Formula (forward) which is a collocation polynomial but in the discrete world, not the continuous world, although it can of course be interpolated. This polynomial is in the index "k" where $x_k = x_0 + k$, so we are on an integer grid. Alternate formulas are given with spacing equal to h instead of 1. There are other forms of this collocation poly that go with names like Gauss backward, Stirling, Everett, Bessel. You use these formulas *to interpolate data* between points where you have data, such as to interpolate a published table, or to interpolate experimental data. The idea can be extended to unequally spaced grid points using "divided differences" in Chapter 9. Numeric differentiation is discussed and is not very accurate. Numeric integration can be done using any of the interpolating formulas, and this usually means constant grid spacing. Here you end up with Trap Rule, Simpson's Rule, and lots of other "rules". These are all error $f^{(n)}(\zeta)$ methods, whereas Gaussian quadrature is MUCH more accurate for the same number of terms, but the cost of course is unequal spacings.

So I have some notes on every chapter from Chapter 2 through Chapter 15 of Scheid. I looked a bit at Chapter 16 on singular integrals, but stopped there.

The **third** section deals with the rest of the book.