RECORD OF SOCIETY OF ACTUARIES
1994 VOL. 20 NO. 2

ACTUARIAL ALGORITHMS

Instructor: DOUGLAS N. HAWLEY
Recorder: DOUGLAS N. HAWLEY

This session will review actuarial algorithms and programming techniques that have proved useful in many applications over the years, including approximations, interpolations, valuation methods, etc. Samples of programs that illustrate how and why the technique works well will be provided. Many of the algorithms discussed will be independent of the programming language used.

MR. DOUGLAS N. HAWLEY: Chart 1 illustrates the Sieve of Erastothenes. There are two algorithms that are probably the most well known and that I've heard of. What is a Sieve of Erastothenes?

CHART 1
SIEVE OF ERASTOTHENES

1	2	3	~~4~~	5	~~6~~
7	~~8~~	~~9~~	~~10~~	11	~~12~~
13	~~14~~	~~15~~	~~16~~	17	~~18~~
19	~~20~~	~~21~~	~~22~~	23	~~24~~

FROM THE FLOOR: Prime numbers?

MR. HAWLEY: Prime numbers, right. Algorithms are generally sort of a mathematical, geometric, whatever, exercise in order, a series of steps, a recipe to get a desired result. What do you do first in the Sieve of Erastothenes?

FROM THE FLOOR: Strike out multiples.

MR. HAWLEY: If you have a list of numbers you pick out all multiples of two. Then cross out all multiples of three, and finally cross out all multiples of five. What you'd have left would be prime numbers.

We have another one—the Euclidean algorithm. If you have two numbers, what can you find with the Euclidean algorithm? The lowest common multiple, or put another way, if you have a fraction, it reduces it to the simplest fraction by finding the highest common divisor.

PARADIGM, METAPHOR AND SIMILE

Because this meeting is comprised of computer-type subjects, I thought it would be good to show a metaphor, a simile, and a paradigm.

Several of the handouts from this session come from articles that were in *Digital Doings*, the Computer Section newsletter. If you're not a member of the Computer

Section, I would certainly recommend it if you're interested in anything about computing or programming. We have a number of good articles.

GENERAL PROGRAM FEATURES

Let's look at some neat features or ways to structure your program. I'm not going to tell you how to do things. I'm only going to make some suggestions. We'll get to more detail on data entry later. Listed below are some features that might be nice to have in your programs. This particular example is a pricing program for universal life. The assumption files. Now typically you write a program, you run a program and you finish up for the day. You have put in a certain set of assumptions. Do you have to start from scratch the next day and reenter all the assumptions?

- Data Entry Screen Menu
- Assumption Files
- Run Program
- Utility Section
- About This Program . . .
- General Help Index
- Run-Related Module

FROM THE FLOOR: No.

MR. HAWLEY: No, you don't want to do that. What do you want to do?

FROM THE FLOOR: Save it.

MR. HAWLEY: You want to save your assumptions from the previous day so you can load them later. So it would be nice to have a feature where you can save your assumptions from one day's work, pick them up the next day, or even a month later. You may have to enter 5,000 individual items to run a program. You'd like to be able to come back and start where you left off, so that's a nice feature to have in your program. Next, it would be good to be able to run the program so it's nice to have a run-program feature. The utilities section allows you certain local features. It does sound effects, but the automatic option is no sound effects because most people hate them.

Let's say hypothetically you have a program that several people might want to run. Wouldn't it be nice to be able to just have your program tell you what it does and a brief statement or a paragraph about the program? Sure, it would help. A short title just tells you the name of the program. So it's good to be able to have something that you can pop up which would describe the program. That is what this particular feature does. Help. You should be able to have help and you should also be able to call up related programs that are associated with the program you are in. In this particular case, you can call up a program and it will calculate universal values, etc.

Some of the program functions, such as data input, are quite detailed. So on the input it would be nice to break those up into separate sections. Maybe you want to go into the program and all you want to do is try different expenses, so this would allow you to just go directly to the expense section rather than go through all the different kinds of options that you have.

Would you like to call up the mortality assumptions and enter the mortality for 100 years while running the program?

FROM THE FLOOR: No, not at all.

MR. HAWLEY: No. What would be a better idea?

FROM THE FLOOR: Use the table.

MR. HAWLEY: You would construct a table once and then be able to reuse it in different programs. Maybe you could run the same mortality table in your whole life program as you're running in your universal life. So you build a table once and keep it. You've probably got some programs out there, too, where you could modify the mortality, multiply it by a factor. Let's say hypothetically you've got your mortality table and then somebody says, we've changed our underwriting. What you're going to have to do now is assume 150% of mortality in the first five years followed by 100% because we're changing our underwriting and we think that would be the way the mortality's going to fall out. Do you want to create a new file at that point? No, you should use the same file and have a set of factors that you can apply to that table, so we put that in. You might have a flat extra mortality. What you're trying to do when you're writing this thing is imagine the future if at all possible. Don't program only the things that you need today; you won't be able to use them tomorrow when the situation changes.

Now in calculations, let's ask some leading questions. I will mostly be talking from a BASIC point of view. To the extent that that applies to the other languages then all these rules will hold. When you're doing calculations and you have a value that will be an integer, what would be a reasonable thing to do with that value for the efficiency of the program?

FROM THE FLOOR: Maybe it's the number of decimal parts.

MR. HAWLEY: In BASIC, you would put a percent sign after the variable. You lose less memory and you do your calculations more quickly. How about APL? Is there a similar sort of thing in APL?

FROM THE FLOOR: There isn't a problem. The program uses the amount of decimals necessary.

MR. HAWLEY: They're all floating decimal point regardless.

FROM THE FLOOR: APL knows what it is.

MR. HAWLEY: So you don't have that situation. And in Fortran you would declare that as an integer, correct?

FROM THE FLOOR: Right.

MR. HAWLEY: And is that the same in BASIC? Do you put a percent sign after it? I used to program in Fortran, but it has been a number of years.

FROM THE FLOOR: Well, yes, but it would be the first letter of the variable.

MR. HAWLEY: If the first letter of the variable is I through N or something, they are automatically integers. The point is, if something is always going to be an integer your program will run fast or use less memory if the program knows it's an integer, however you do that in APL, BASIC or Fortran. As an example I wrote a program, changed compilers and the run time was twice as long. Of course, this was several years ago, so it was running on an XT or an AT or something like that, so run time counted at that point. I cut the run time down by 30-50% just by going through and seeing that all the integers defaulted to floating points. I changed them to integers and cut the run time way down. Now this would vary by what language you're using, but it can't hurt if you are to make sure that an integer is treated as an integer.

On the other side, I've written programs for immediate annuities and GAAP factor programs where you're discounting, at least in the old days, at 15%. You're going out maybe 100 years with interest and survivorship. In BASIC, if you use a default floating point at six or seven decimal places, it turns out in some of these calculations I would be ending up dividing by zero at the extreme ranges, because, after the discount for mortality and interest, we'd be getting down to a zero divisor, if we used single precision. In BASIC, you then go to double precision, which is the variable followed by a pound sign. The way you usually find out that you do need to go to double precision in order to get the accuracy is by making a mistake the first time around and getting zero divides or unreasonable numbers. Also, commutation functions is another place you may need to go to double precision. Now what do you do in Fortran when you need more accuracy?

FROM THE FLOOR: Use double precision.

MR. HAWLEY: It's the same thing? You declare a double precision or you use just the declare statement?

FROM THE FLOOR: Yeah, I think we define double precision.

MR. HAWLEY: Does APL need to be set to double precision as well?

FROM THE FLOOR: We get 15 decimal points automatically.

MR. HAWLEY: Automatically?

FROM THE FLOOR: You can globally set the precision for everything.

MR. HAWLEY: Most people use ASCII files for input or output from various different programs. Now let's say you're working on a select mortality table. Would you still be using an ASCII table? I know what you do in BASIC. I don't know what you do in your other languages. Let me put it a different way. What's a reason not to use a sequential ASCII type file? Let's say you've got select mortality and you use a sequential file. You read in all the values because you can't pick out the one you want in a sequential file. You've got to read everything or you don't read anything. Then you put all these things into the memory, but you're taking up memory you don't need. In BASIC you write a random access file. You pick out the pointer that

you want. So, for example, in this mortality file, if you have hypothetical mortality from ages 0 to 70 by five, and you're running a profit study for age 30, you can write your program to read out just the mortality for age 30 when you run a program. You don't have to read all values. You just read in the one you want when you run the program. Again, what would you do in APL in a situation like that or do you always read in all the values? Do you have the option of picking out a particular record out of a file?

FROM THE FLOOR: Usually, anything is variable in the program. Very rarely would I use files.

MR. HAWLEY: Okay.

FROM THE FLOOR: If you use files you can pick out this one item.

MR. HAWLEY: How about Fortran? I think Fortran is probably close to BASIC in this regard. The point is that you have different kinds of files that you can use in your programming; so determine what kind of files you need. You should structure your program so you don't have to do any more file reading than is necessary.

Output. Does anybody write any programs anymore that have printed output only? (One hand raised.) I think every program has printed output as an option. What are your other choices?

FROM THE FLOOR: Files.

MR. HAWLEY: What else?

FROM THE FLOOR: Screen.

MR. HAWLEY: Yeah, print to screen. I guess I'd call print-to-screen output files. It depends on the use. If you're running a valuation report print to screen it may not be useful. Somebody has to read the printout. How about pricing? If you're running a pricing program what would be a reasonable default: screen, file, or paper?

FROM THE FLOOR: Paper.

MR. HAWLEY: Why do I want the screen?

FROM THE FLOOR: So you could find out if you want to save it.

MR. HAWLEY: Look at it and find out what you did wrong. Find out that you set the commissions at 30% when you meant 300%. A tree can live for another day. File output can be used to review later. In the case of a pricing program I always use print to screen first; if it's reasonable and if I like it and I need to save it, I print it out. Otherwise, I flush it and go onto the next job.

Output and files. If you're running Lotus or other spreadsheets, I suppose it would be good to have an output that would be compatible with it. I don't do that, but some people might. A show of hands for anybody who does that? Several people do that

and I've talked to many people that do that, so depending on your situation you may want to have a file output that's compatible with some worksheet.

And what special features might these programs have? I think we've covered some of them already. It would be nice to be able to retrieve help. When you get to the section that says mortality, you could call up something that says this means mortality per thousands. You might like to learn about your program while you're not actually in front of the computer. It doesn't hurt to be able to access help while in the program and have paper documentation.

One of the things that may have happened to some of you is that you've got somebody who writes a program and knows perfectly well how to run it and how it works, but that person doesn't stay at the company forever. Documentation should not be regarded as optional but as an essential part of programming.

Many people like graphics. If pie charts sell at your company do it. Who does like graphics and who does all the graphic stuff with the pie charts and the bar graphs? Does anybody want to say it's a neat idea?

FROM THE FLOOR: It can come in real handy for illustrating your point to somebody who doesn't know what you're doing.

MR. HAWLEY: For an executive summary?

FROM THE FLOOR: That's another way to put it.

FROM THE FLOOR: Actually those pictures can really scare senior management more than the numbers.

MR. HAWLEY: You mean the bar that goes way, way down below the x axis?

FROM THE FLOOR: You just wouldn't believe how well it works.

MR. HAWLEY: Yeah, the words "negative $10 billion" aren't as impressive as a bar that goes off the bottom of the page.

FROM THE FLOOR: Right.

MR. HAWLEY: The other thing that goes with this is graphic user interface (GUI). I think GUI is something you don't want to step in, but other people feel differently about it. I believe GUI is not necessarily a good thing and, again, this is a matter of taste. You can fight with me on that, if you like.

The other thing that would be helpful could be suggested by a hypothetical example. Let's say that your boss says he wants these pricing studies run with 30 variations on the lapse rates. I'll propose a scenario to you and see if you can suggest an improvement. Scenario A: you make up your lapse rate; you run the program; you compile the results; you make up another lapse table; you reinput the assumptions; you run the program; you put the results somewhere, and do this 30 times. Can somebody suggest an improvement?

FROM THE FLOOR: Run batch.

MR. HAWLEY: I may be using batch in a slightly incorrect fashion, but I'm suggesting that you make up a set of assumptions. Don't run the program—make up another set of assumptions, but don't run the program. Do that as many times as is needed and then include a feature which allows you to queue up these sets of assumptions. The program runs fast enough so that you can't do anything else. On the other hand, you're wasting your time sitting in front of the computer while it's running. Don't do that. Make up all the assumptions and then go away. Let the program run all the different types of assumptions without interfacing it.

PROGRAMMING DOS AND DON'TS

Connectivity. Let me suggest another scenario. Determine if you can improve on it. You want to run a traditional life pricing program and you have to create the reserves first. So you run the reserves, get your printout, and then code those reserves into your pricing program. Improve on this scenario for me.

FROM THE FLOOR: You could have the original program create a file.

MR. HAWLEY: Yes. It would be really dumb to have a printout to paper and then code the numbers into something else. What you'd have to do is create a file that you can run into your next program. Similarly, after you've run the pricing program, let's say that you want to run a model. Do you take the results of your pricing program off a sheet of paper and code those as input into your model office?

FROM THE FLOOR: No.

MR. HAWLEY: No, you do the same thing. You have a file of output of the pricing results. You put that into your next program as a cell. So the ideal situation is to have programs that are connected from beginning to end. I know of situations that occurred in the old days of APL. An APL program was written to get reserves for settlement contracts; then those numbers were coded into another program that did the reserving. I think the story was that they couldn't create a program large enough because of the space limitations to do both jobs. Simplicity's always a virtue.

Structure. In programming I used to get in big trouble because I wrote the program and all this stuff went right down there without any indentations and I had line numbers; this seemed to be a huge sin. So what do you do? You're writing your program. Do all your appropriate indentation, and do your commenting. Does anybody have anything to say about program structure; are there any enlightened comments? Is it important in Fortran? Do you do the same sort of things in Fortran as you do in BASIC, like indentations or loops? You would do that. I think that was an affirmative back there from Fortran. Now how about APL? Is there a similar sort of paradigm in APL programming? Any sort of indentation or particular structure for APL?

FROM THE FLOOR: No.

FROM THE FLOOR: If you get the latest version.

MR. HAWLEY: The latest version has that sort of thing? As I recall, the APL fanatics that I knew would say something like, "I wrote this 500-line program in one line."

User library. I'll talk from the BASIC perspective again. My stunt programmer has many neat input routines so he can do one kind of input statement for a file, one kind of input statement for a string variable, one kind of input for integers, and one kind of input for floating point. He has various routines, for example, something to give error codes. If you make an error it will actually show the book description of the error. Another example would be if you're writing something with many annuities. This would not be a subroutine in BASIC, but it would be a user-defined function. If you have a program that's going to calculate annuities 50 times, the idea would be to have a user-defined function that would calculate the annuity given the variables rather than redefining the calculation 50 times. Wouldn't the same apply to Fortran, user libraries, user-defined functions and that sort of thing?

FROM THE FLOOR: Yes.

MR. HAWLEY: Parallel systems. A company which will be nameless had various accident and health (A&H) programs which purported to do loss ratios and things like that. The reports went to different people, but the reports could not be compared in any way. There was no way you could take one report and make it match another report, because different databases were used. One of them had a user interface so you could play with the numbers as an intermediate step; the other one didn't have a user interface. So one set of people would be looking at one report, and another set of people would be looking at another report. These were supposed to give the same answers but had different numbers on them. It is not a good idea but things happen and there are always historical reasons for them.

In the same company, the general attitude within the actuarial department was that EDP was worthless. This caused the actuaries to write programs that they shouldn't have written. There was an administrative system for a block of business. They couldn't or wouldn't get anybody to write the reinsurance section. Can anybody guess what happens next? What do you suppose they did? They wrote a separate system that was supposed to be parallel to the administrative system except it only did reinsurance. So there would be issues for main system and issues for reinsurance. There would be lapses for the main system and lapses for reinsurance. Does this sound like a good idea? It gets even worse. One morning, there were not many smiley faces. The reinsurance disks had been erased. Do you have a relatively negative relationship with the EDP/MIS (management information system) area? I don't really know what you can do about this, but I feel this is one of the most important things that goes on.

I feel very strongly about a number of issues. This will be nontechnical so you can knock me down on the relevance to actuarial algorithms. I think that the actuaries often abdicate their responsibility to communicate with EDP, half the time the EDP department is relatively incompetent. I have seen so many cases where an actuary told the EDP department to calculate a yield rate and six months later EDP hasn't done it. Well, of course, yield rate will mean nothing to your EDP department. The actuaries will refuse to give precise specifications. They just say go off and do it or

won't demand the proper priority. EDP drops the ball with actuaries' assistance. Everybody's mad at everybody. The project falls apart and then the actuary does something really insane like write a parallel reinsurance system to the administrative system. Don't let this happen.

I guess the first thing closest to home is, to the extent this is appropriate, reform yourself. Make sure that you can talk to your EDP guys. Production jobs shouldn't be written by actuaries unless they are under the auspices of MIS/EDP. Actuaries often write programs that aren't documented. They don't have the same standards. Somebody else comes in and uses a different language than the actuary that replaces him or her. I think it's a bad idea for actuaries to write production valuation programs that are run every month. One shot deals, like experience studies, pricing, or modeling is fine; these are run irregularly within the actuarial department. Something that involves valuation for the company is generally a bad thing for an actuary to write.

Good relations are needed with the EDP department. If you think that you are really righteous and giving good, valid specifications to the EDP department and they're being unresponsive, see about changing EDP. Go do something about it because giving up on the EDP department and doing it yourself is a lose-lose situation. If they have a job and they're not doing it, first make sure that you're communicating with them as well as possible. When you give up on that point, do something about changing the team.

Hardware. Now we actually have some people at the mini computer table. I know nothing about that. Is the mini computer area generally thriving, decreasing, or are they picking up as much from mainframes as they're losing to micros? What's happening?

FROM THE FLOOR: I've been using a mini since 1981.

MR. HAWLEY: Okay, what about mainframes? Is anybody here representing mainframes at all?

MR. CHRISTOPHER P. KEENE: We're writing an administration system on the mainframe. I think of it as being the last mainframe administration system. Micros allowed people to do their own thing for a long time but now EDP has discovered a way to get back control and allow these crazy (they think) people to do their own thing. That's the whole idea with LANs and Client Server and everything. They're now billing us for doing things on a LAN, so, they're back in power in EDP.

MR. HAWLEY: I'm sure there's going to be a great deal of politics involved. I would say, regarding hardware, if you're in the PC or mini environment, there's no reason to have anything less than a 486. They're dirt cheap. They're so much faster than anything else. On the Pentium/586 level, you're still going to pay quite a bit of a premium for a while, and I don't know that they're that much superior to a 486 from what I've heard.

Put comments in your program. They don't take up any of your execute code and someone will probably be trying to figure out what you did. So even if you're the only one that's going to ever look at the program, put in the comments.

Files. When writing a program, the whole thing is constructed with variables. You have variables sitting out there taking up memory. You don't have a file unless you have an output file. I wrote a program for cash-flow testing, and one of the things it does is inputs an asset file. When dealing with smaller companies, you may have 200 or 300 assets. Each individual asset takes up about 130 bytes. Let's say you didn't want to limit the number of assets. Let's say you've got a company that may have thousands of assets and you're storing these things as variables. Let's say you're in an environment where your maximum memory available is 620K, and that your program takes up 300K. What will eventually happen?

FROM THE FLOOR: You'll be out of memory.

MR. HAWLEY: In my case somebody will call up and say what does error 7 mean? Error 7 means you just ran out of memory. Now if you don't want to limit the number of assets what option do you have with these assets as you go through and process them?

FROM THE FLOOR: Summarize.

MR. HAWLEY: Let's say that you're not doing any modeling. Let's say that you want to keep it on an individual basis for every single duration.

FROM THE FLOOR: Write a file.

MR. HAWLEY: Write a file. So my solution was to take the input file, read everything out of it, and write it into a new file. As I go through the durations, what happens to some of the assets in each duration as you go through the processing?

FROM THE FLOOR: Sell off.

MR. HAWLEY: Sell off, mature, get calls, whatever. Each time you go through it, you write a new file and one hopes it will even get smaller. But here is the way around it. My knowledge was limited to an extent so this didn't even occur to me for quite a while. In a situation like this, it is not necessary to write things to variables. You can write temporary files. The other thing you must remember is to erase all your files after you're done or you will soon fill up your machine with files and you won't know what they are. It happened to me.

Does anybody know what OOPs is?

FROM THE FLOOR: Object Oriented Programs.

MR. HAWLEY: I take a different view of OOPs. OOPs to me means that I wrote a program once and never wrote a second program, because I just salvage everything I can out of the first program. If there's a mortality input, do I write mortality input again? No, I've already written mortality input. OOPs to me means that you do not

write more new code than you have to and it turns out you don't really have to write that much new code. If you just save all the old stuff and adapt it and if you have a routine that inputs a file why is it a problem if it's a different file? Don't change any more than you have to. Pick out the code and use it again. It saves so much time. This isn't news to anybody is it?

What if I start to write a program and something interrupts me. Or I just have no idea how to proceed, so I go away. A week later I might know how to finish it. Sometimes, if I try not to think about something, I can do a better job when I come back to it. Another thing that works for me is having two or three things going on at once. If I have one thing going on, I won't be as efficient as if I'm trying to do two or three things because they kind of bounce off each other. An idea that applies one place may turn out to apply somewhere else. This is what works for me. Sitting down and doing eight hours of programming is much less efficient than having two or three things going on, working for an hour, and going away for three hours.

Where's numerical analysis on the syllabus these days?

FROM THE FLOOR: It's in part three.

MR. HAWLEY: Let's quickly talk about some stuff and some real life examples. By the way, I would like to credit Sharon Hawley with the charts. They were done I think with Paintbrush.

If you're approximating by recursion you have many different things to worry about. Will you actually find the solution? Of course, there are occasions where you have a complex solution to your problem. You've got the case where your series does not converge.

What do you do with bifurcation? (See Chart 2.) You want to find a place where you solve for zero. Find a place on your axis. First, you find an interval where the function is positive one place and negative the other. In step two, you cut that interval in two and pick out at least one where it's positive at the one end and negative at the other. You keep doing that and you get closer and closer to that point where it crosses the axis. This is not real quick, but it's simple and it works.

Now after I wrote an article about the Newton-Raphson Method, somebody wrote back and pointed out that there is probably a more efficient method, from a programming point of view, than the Newton-Raphson. It actually takes more steps but you have less work to do in each step. He suggested that the Secant Method was a better approach to solving for functions or solving equations for 0. Chart 3 shows the Secant method. You select two points. Between these points, the function is positive at one and negative at the other. You draw a line between the two points to find out where that line crosses the axis. That's your next iteration. You can see you're already close at the first step. Now, of course, this is a nice function.

The other approximation is the famed Newton-Raphson method (Chart 4). Instead of using the secant line between the two, you use the tangent line or the line defined by

the derivative. The thing that's neat about the Newton-Raphson or the Secant methods is that if you can draw the picture you know the formula.

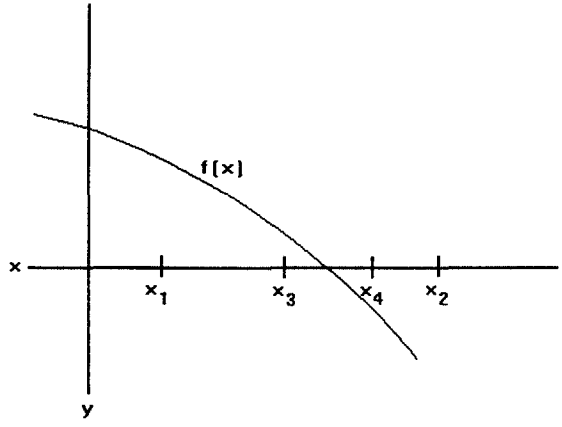
Where are some applications of these models? Here are the things that I run across: Linton yield rates or ROI calculations. Essentially that is a polynomial to the *n*th degree where *n* is the duration of the profit study or whatever. It's the *n*th year. So if you're talking about a ten-year Linton yield, you'd be solving a polynomial to the tenth degree. If you have a profit study for 30 years, and you want to find out the average ROI for 30 years, you're solving the polynomial of the 30th degree.

Okay, math fans out there. For what kind of polynomials can we find exact algebraic solutions?

FROM THE FLOOR: Up to 4 degrees.

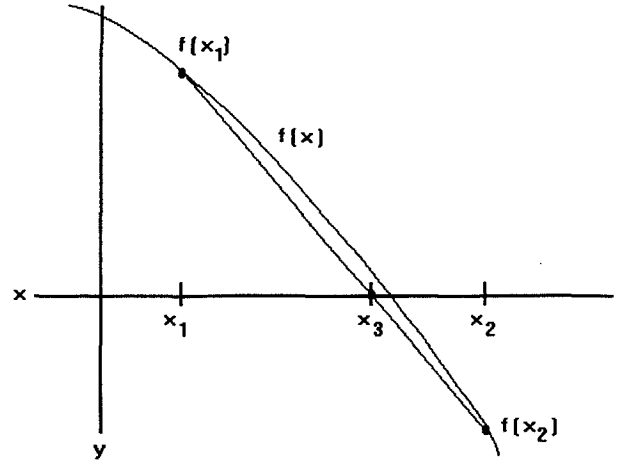
MR. HAWLEY: Up to 4 degrees. There's nothing algebraic thereafter. So we know that we can't, in the general case, find the solution to a fifth degree or higher polynomial, so we have to go through these numerical methods. Now where do we get those? We get these in the ROI, in a profit study or if you need a Linton yield rate, use the same process. Dennis Radliff wrote a paper on universal life target premiums which gives alternatives to what I did. When I did the universal life target premium and such calculations, I went through the monthly calculation, doing the approximation, doing the second approximation and so on, four or five times.

CHART 2
APPROXIMATION: BIFURCATION



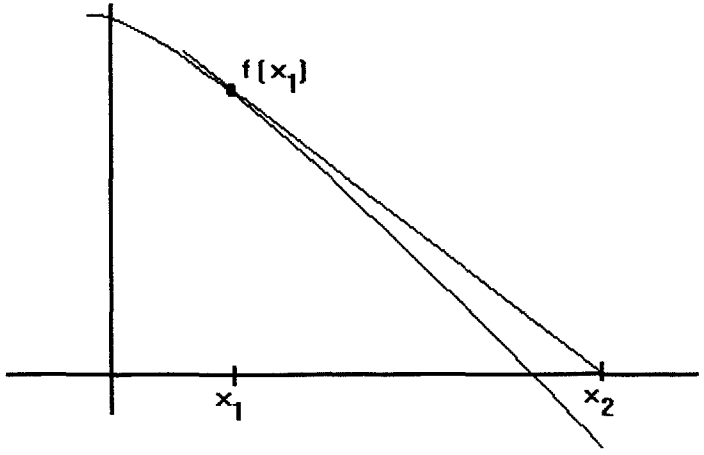
Pick middle of interval where *f(x)* changes sign

CHART 3
APPROXIMATION: SECANT



$$x(n+1) = x(n) - f(x(n)) / (f(x(n-1)) / (x(n) - x(n-1)))$$

CHART 4
APPROXIMATION: NEWTON-RAPHSON



$$x(n+1) = x(n) - f(x(n)) / f'(x(n))$$

Calculating a universal life for 95 years is a complex process and takes a while even on the new computers. Sometimes you'll need to do this. I think Dennis mentioned that there are some cases where you can't go year by year which was what Dennis did as an alternative to doing the monthly calculations. He has an annual shortcut. In some cases where you have some of the older policies where they have split interest rates (one interest rate on one level, one interest rate on another), there are some other things that work as a general annual solution. So if you're working at this thing generally and you want to solve it for a target premium or something like that, you probably in the general case are going to have to go to a Newton-Raphson or some other approximation. However, Dennis Radliff has a contribution to *Digital Doings* (Computer Science Section Newsletter), July 1993 issue, "Speeding Up Universal Life Monthly Calculations", that shows you how you could do it the easy way in most cases and just get your universal life values on an annual basis.

In our last charts we actually have a source code for the calculations of the constant prepayment rate (CPR). We will save those programs for later. Most of the things I wrote up for Newton-Raphson will apply to other approximation techniques. They have a good first estimate. So, for example, if you're solving for return on investment in a program, you shouldn't do it at all if all the returns for all years are positive because you can't get a real solution. Find out whether or not you can get a solution. You should get a good first estimate. When I'm solving for an ROI, I make a simple assumption: the profit is equal after the first year. The loss in the first year is followed by a number of equal profits. I also assume that the interest rate is the average interest for durations between one and the end of the projection. Now, obviously, this is a the wrong solution, but this will get a first estimate of the ROI. Use that as a simplification. The better start you get, the quicker you converge to a good answer. So look at your problem. You can make a guess that isn't going to be the right guess, but the better guess you get with the first step, the quicker the process. Again, make sure the variable type is appropriate as you get close. I've had situations where I was using single-digit precision, and I would get an answer that would kind of bounce around and would, in fact, after a while get away from the right answer instead of getting closer to the right answer. I don't know the theory behind all this, but in some cases you want to go to double precision on these approximations.

Make sure that the accuracy improves with any step. Again, I've had situations where I looked at what was happening, and the answer would get farther away. So I'd put in a test. If the answer got farther away I'd do other things rather than just take the normal next recursion. We'd have some special logic in there to make sure that the guess would get better at each step. Run a test to make sure that you can get a solution.

The other thing is that you may run into a bizarre situation. You might get into an infinite loop. I'm not going to make more than ten guesses regardless of their accuracy. Don't just keep going until you find the right answer. You may never find the right answer for whatever reason.

SHORTCUTS/LONGCUTS

Pricing. How many people have either written, run or worked with pricing programs? (Most of the people raised their hands.) How many do modal pricing beyond just

plain annual? I guess at least half the people are doing some pricing with modal stuff. In the past, if somebody was doing modal pricing (let's say he or she wanted to go to quarterly), he or she wouldn't run profits each quarter. The program would make an approximation to the various values on a quarterly basis. Chart 5 is an example of how you get the modal approximation to surrender value in different situations. Then you do your modal approximation of premium, your modal approximation of death benefits, etc. So you write your program, but you have all these modal approximations that you run only annually. Why would you do that? It's faster. If you're running monthly you get 12 calculations per year. So you do that as a matter of speed. Now there's still some validity to that in that this would be faster if you're only taking one pass for each duration rather than going through quarterly.

Now I've gotten to the point where I am combining all the modes. So if you're doing pricing you can say I'm doing 25% annual, 25% semiannual, 25% quarterly and 25% monthly. Why? First, I'm not smart enough to do approximations like this. Second, with the newer policies (let's say it's universal life) can you make an equation like this? Would this same equation be valid? The answer is no. Why is the answer to this not valid for universal life? You don't interpolate cash values in a universal life policy as you do in a traditional life policy. With universal life, you can pay off more, and you can do all these bizarre things. It isn't a simple interpolation. You actually do a calculation for each month unless you can figure out how to do it annually. In my case, I was calculating universal life values monthly and they, in fact, are not a linear interpolation between the annual values. So the interpolation solution for the cash value is no longer valid. Second, my program is already going through and calculating out monthly values. Why not just calculate all the values monthly? My approach is to do every single calculation monthly—monthly death benefit, monthly surrender value, etc., and then just sum them up for a year. So rather than doing an approximation like this for the surrender value, I calculate 12-month surrender payments, add them up, and that's the surrender payment for the year. The same thing applies to death benefits.

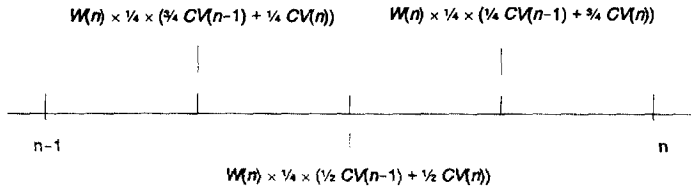
Now what makes this feasible? This was a more questionable thing five years ago. Why is it more acceptable to do it monthly?

FROM THE FLOOR: We have quick runtimes.

MR. HAWLEY: Fast machines. Very good answer. The time is not so relevant. Doing this thing monthly could mean the process takes 30 seconds instead of 20 seconds which is not that big a deal. Years ago it might mean five minutes instead of one minute. So the time factor is no longer so important. I guess the clue is that we adapt to today's technology. The validity of this kind of approach was the time element and if the time element is no longer as relevant, then you have to ask the question, are these sort of methods relevant? They may be, but you should be asking the questions.

CHART 5
ANNUAL APPROXIMATION TO SURRENDER VALUE PAID
QUARTERLY MODE

Annual Lapse - $W(n)$
In Force = $IF(n)$
Cash Value = $CV(n)$



$$\text{Cash Value Paid} = IF\ (n-1)\ \times \sum_{p=1}^{4} \tfrac{1}{4} \left[\frac{4-p}{4}\ CV\ (n-1) + \frac{p}{4}\ CV\ (n) \right] \times W(n)$$

$$= IF(n-1)\ \times\ W(n)\ \times \left[\frac{7}{16} CV(n-1) + \frac{9}{16} CV(n) \right]$$

Generally for m payments annually:

$$\text{Cash Value Paid} = IF\ (n-1)\ \times W(n)\ \times \left[\frac{\frac{1}{2}m^2-1}{m^2}\ CV(n-1) + \frac{\frac{1}{2}m^2+1}{m^2}\ CV(n) \right]$$

- Ignores death benefits and early lapses
- Assumes cash values are interpolated

MONTE CARLO TECHNIQUES

In the confines of my own office, I call Monte Carlo techniques sadistics. I'm not really proficient in this area. However, one of the things that we need to know more about is how to use Monte Carlo techniques. What are some examples? I have no idea of the exact detail but I'll give you some ideas. In the example below the program is trying to calculate the probability given a small finite population of let's say five or ten people of a given order of death. Now if you start off doing a kind of branching diagram with the probabilities and you follow through and you pick out the cases where age ten dies before age five, but after age six, it gets ugly fast. In this example, I went through one branching and two ages. Imagine how ugly that calculation is. Now maybe somebody really clever can write out the solution for that by actually doing the long version of the calculation.

MONTE CARLO VERSUS USUAL APPROACH TO MULTILIFE

Usual			Monte Carlo	
Age 1	L	L	**Trial 1**	
		D	Age 1	L L . . . .
	D	L	Age 2	L D . . . .
		D		
Age 2	L	L	**Trial 2**	
		D	Age 1	L L L . . . .
	D	L	Age 2	L L D . . . .
		D	.	. . . . .
			:	: . . . . .

FROM THE FLOOR: At age one, why do they continue to branch after he dies?

MR. HAWLEY: Branches should terminate upon death. If you actually went through this calculation, and went through the logic of this whole thing, it would be a long, long time before you reached a solution. So what do you do instead? You run Monte Carlo trials. You take a guy age one and do a random number generator. The probability of the guy aged one dying in the first year is 0.001. You essentially flip a coin with your random number generator and you have turned into a deterministic scenario in which he either actually dies in that trial or he actually lives. That's a given trial. You run these things out to the end, until in each case the trial ends with a last death. Then you just count up the number of trials in which a given order dies. So if you run 10,000 trials and 300 of them result in the thing that you're looking for, then your probability is 300 over 10,000 or whatever numbers are used in the example. Again, I'm not an expert at this. There is an article, if you speak or read APL, and if you want to actually look at the program. Rob Foster, the author, is at Integon. I assume that he would be happy to discuss it with you. There's another reason to join the Computer Section for people who haven't already.

Another way you could use the Monte Carlo method. I don't have any kind of theory; this is gambler's ruin applied to insurance solvency. Let's take an overly simplified case. Let's say that in any year, you're 25% likely to make profits equal to 10% of your assets and 25% of the time you'll make 5% of your assets, 25% of the time you'll make 0%, and 25% of the time, you'll lose 5% of your assets as a bottom line. So your question is, what happens at a given surplus level? What are the chances in 20 years that you'll be broke? So it's a modified gambler's ruin kind of problem. Again, try to write out the long formal solution to that using strict probabilistic arguments. It's not a fun thing. Run a number of Monte Carlo trials and, again, see how many fail and how many survive after the 20 years. This is a little bit more complicated than one sentence, but it's not a complicated idea.

Reinsurance. Is anybody here in the reinsurance area? Oh, good. Do you work with these Monte Carlo techniques in modeling for reinsurance or stop loss?

FROM THE FLOOR: No, not a lot.

MR. HAWLEY: Okay.

FROM THE FLOOR: What if you're retention setting?

MR. HAWLEY: Retention setting or catastrophe reinsurance would be an area in which to use the Monte Carlo techniques. I would say it applies more to the stop loss or catastrophe, but to some extent it's useful for the retention level. So there are many applications in reinsurance. You can also do this if you want to see a catastrophe level of mortality. If you want to go through your in-force business and look at your distribution of possible death benefits, again, by actually working out the calculation with 100,000 lives, it's ugly. Don't do that. Run your Monte Carlo and run out your 10,000 cases and see the various different levels within the 10,000 cases. Do your distribution that way. I don't know how many trials you have to run. I don't know statistics; you'll have to look it up.

SERIES

They wanted these notes 45 days in advance and I have subsequently run into something which is an excellent application. Previously, I racked my brain to find a practical example of using series in actuarial work and the thing is you don't use a series for e^x. You'd say e to the x. And you don't use a series for sine. You say sine. You don't do hyperbolic tangent. I mean these things are really series in terms of the way the machine language or the libraries or whatever look at them, but we don't care. Subsequently, two ways came up which I could have included in the notes. I ran into an actual case study where I have a series situation, and I'll try to move right along and get to that.

We have a program for some of your series written in C by the fellow who literally wrote a book, if not, *the* book. He used to be my boss when I was a teacher. I believe he's now at DePaul in Chicago and writes books in C. His name is Richard Johnsonbaugh.

CASE STUDY/WORK-IN-PROGRESS

Problem—generate p "random" interest rate scenarios for cash-flow testing.

Assume—log normal interest distribution according to input mean and deviation, and "drift" is derived from current long-term rates.

Input—current interest rates for a variety of risk-free bonds, and the sort-term interest rate volatility/standard deviation.

Calculation:

1. Use numerical analysis and price/interest relationships for the various bonds to solve for "spot" interest rates for all durations.

2. Generate p random numbers from 0 to 1.

3. Approximate F, the cumulative normal distribution by Taylor series.

4. Solve for the inverse of F at the p random numbers by recursive approximation techniques.

5. Answer from above, along with volatility and mean give unadjusted short-term interest rates.

6. Adjust rates for "drift."

7. Repeat steps after first to get adjusted short-term rates for all durations.

8. Use short-term rates derived from above rates to solve for other bond and mortgage rates required for model. Again use numerical approximation and interest relationships in order to calculate interest on other assets.

Let's go on to the case study/work-in-progress, and I think this is kind of neat because it's what is happening in actuarial work, and it touches on many of the things that we're talking about here. How many people are looking at cash-flow testing in which you need to generate interest rates? This is a very current topic. The usual assumption is that you have a log normal interest rate distribution which, as I understand it, is that log of the ratio of the interest rates is normally distributed. So rather than taking a value from the distribution, you take e to that value from the distribution because it is log normally distributed. To some extent, this is an empirical kind of thing because if you just assumed interest rates were normally distributed, what ridiculous thing would occur some of the time?

FROM THE FLOOR: You'd get negative interest rates.

MR. HAWLEY: You would get negative interest rates. As a practical matter, we don't assume a normal distribution. Perhaps it's because of empirical information derived from past history. The usual assumption is to say that the interest rate scenarios are log normally distributed. As I say, this is a work-in-progress rather than a finished project, so there may be changes.

Start with your current interest rates as the jumping off point. Look at your short-term rates: one-year, quarterly, semiannual, and so on. For my purposes it's one-year interest rates. You start off with that, plus the scenarios for T bills or T bonds up to 30 years in order to calculate spot rates for the future. You use not only the interest rate, but also the volatility or the standard deviation assumption for the interest rate to run through this process.

Use a numerical analysis or project interest relationships for the various bonds to solve for spot interest rates. Now what does that mean? Let's say you know the interest rate for a five-year bond. You know the interest rate for a one-year bond or a one-year bill. The problem is to get two-, three-, four-, and five-year spot rates.

You use some numerical analysis, plus the givens to solve for these spot interest rates given those assumptions and you work on that. Let's say you have one-year, five-year, ten-year, and twenty-year rates. You have to find all years in between. Your five-year bond rate isn't the five-year spot rate. So you must go from these various different current asset rates into spot rates for however many years into the future.

This is a numerical analysis problem where you may be using a Newton-Raphson Method. You might have to also look at your price/interest relationship to solve this problem.

Then you generate *P* random numbers from 0 to 1. *P* could be 100; it could be 1,000. It depends on how many interest rate scenarios you want to generate. You approximate *F*, the cumulative normal distribution by the Taylor series. I assume that's what I'm going to do at this point. If you remember your normal curve integral from negative infinity to *X*. It's one over the square root of two pi, and *E* to the minus one-half *X* square or something like that?

FROM THE FLOOR: It's something like that.

MR. HAWLEY: Okay, look it up. It's in statistics textbooks. What interesting fact do we not know about that integral?

FROM THE FLOOR: We don't know antiderivative.

MR. HAWLEY: We don't know the antiderivative. The beauty of the situation is this makes a great case study. While I was hunting around for things for power series, my real problem was that I had no problem. Now we have a problem because there is no nice form for the integral of the T square minus whatever it is. So what do we do with it? You make a Taylor series for it. One thing that helps is when you're integrating a function, how do we differentiate the integral of a function? What is the derivative of the integral of *F*?

FROM THE FLOOR: *F*.

MR. HAWLEY: *F*. But we have a big jump start right there. Do your Taylor series for this. How do you know how close you are to the correct answer by taking *n* steps in a Taylor series?

FROM THE FLOOR: Look it up in the book.

MR. HAWLEY: You look it up. The Maclaurin series gives you the error by taking the first *n* terms, the error between the actual function and the series is the last term evaluated at some point.

Let's say you're picking out your random numbers from 0 to 1. Imagine, if you will, the accumulative normal curve. It starts out here on the left at what value?

FROM THE FLOOR: At 0.

MR. HAWLEY: It's very close to 0 and it's going up. Imagine a normal curve. My right arm is approaching one asymptotically and my left arm is approaching zero asymptotically. So I'm picking out my random number between zero and one. Let's say my random number is *R*. I set *R* equal to the integral from minus infinity to *X* as a normal series which I throw out and put in the power series. Then I pick out *M* terms where *M* terms are close enough depending on Maclaurin. I set *R* equal to this power series. I know *R*. I have to solve for *X* in the power series. I have to know

that I have enough terms in the power series to be sufficiently accurate. So then I came back and what did we do before we have a polynomial and M terms. What do we do to solve that?

FROM THE FLOOR: Use Newton-Raphson?

MR. HAWLEY: Use Newton-Raphson or the approximation technique of your choice. So we have a power series. We have Newton-Raphson, and we have a Monte Carlo method.

What I found out so far in looking at this random scenario generation is all of the approaches that I can understand are illogical. After you've generated 100 answers for the second year, you solve for the spot rate for the second year. You have to adjust all these answers so that they average to the spot rate. Now what this implies, and this is ridiculous, is that the second year spot rate is actually going to be what was implied by the interest rate structure a year ago. What the interest rate predicted actually turns out to be true. Do you think that's going to happen? I don't think it's going to happen but that's the way we do these things. I don't make up the rules. This is the adjusted rate. This is all just to get year one. Then more fun. Now what do you have when you've done all this. We have spot rates for n durations. Is your insurance company going to invest all its money in one-year bills in the next 20 years? I don't think so. So what we have is not sufficient at this point. What do we have to do?

FROM THE FLOOR: Calculate the risk curve.

MR. HAWLEY: You have to calculate the risk curve using these as spot rates. So then we're going back to the price/interest relationship. We must do some numerical calculations to come up with mortgages. With the mortgages, it's not that easy because we probably are assuming some sort of prepayment rate that makes these things even uglier. This is a real-life situation where you have all these things going on simultaneously.

VENDORS

There is a law of vendor aversion. I have found that companies least like the vendors they know the most about. This is perhaps counterproductive. I've seen so many people say, "I won't ever talk to that vendor again. I want to get taken by somebody completely new." I would just suggest that getting taken by somebody new is not better than being taken by somebody you already know. Believe what you see. It's always good advice. If somebody says they have something, ask where it's installed and ask if it is working now. Try to get beyond your past experience and problems and think about things that can go wrong that haven't already gone wrong. Obviously, watch out for overruns, time and money, and unrealistic promises.

REFERENCES

I went to a Barnes & Noble book store. Somebody pointed out that most book stores have more books on astrology than they do on science which is unfortunate but true. However, at the Barnes and Noble in my neighborhood they have about 100 or so math titles. If you're interested in following up on any of this and spending a great deal of money, below is a list of some books that I saw at Barnes & Noble and a short description of the book.

Corman, Leigerson and Rivest. *Introduction to Algorithms.* MIT Press—McGraw Hill, Cambridge, MA, 1990. (Big, technical, includes "psuedocode.")

Duncan, R., and Howard Raitts. *Games & Decisions.* Dover, New York, NY. (Technical)

Edwards, Dilwyn, and Mike Hamson. *Guide to Mathematical Modelling.* Boca Raton, FL, 1990. (Practical advice, mostly physics.)

Hamming, R.W. *Numerical Methods for Scientists & Engineers.* Dover, New York, NY, 1987. (High level, lots of stuff.)

Harel, David. *Algorithmics: The Spirit of Computing.* Addison Wesley, 2nd Edition, Redding, MA, 1992. (Good and practical, includes code.)

Hildebrand, F.B. *Introduction to Numerical Analysis.* Dover, New York, NY, 1987. (Text-type book on subject topic.)

Paulos, John Allen. *Beyond Numeracy.* Random, New York, NY, 1992. (Lots of popular topics. Nontechnical.)

Sedgewick, Robert. *Algorithms in Modula-Three.* Addison Wesley, Redding, MA, 1991. (M–3 is programming language.)

Stewart, Ian. *Another Fine Math You've Gotten Me Into.* W. H. Freeman & Associates, New York, NY, 1992. (Funny chapter titles, graphics, probability.)

Wagon, Stan. *Mathematica in Action.* W.H. Freeman & Associates, New York, NY, 1991. (Solving problems using mathematica–Prime numbers, recursion, graphics, number theory.)