

SCPCDOC -- This file contains the Preface, Contents and an example chapter of a book by J C and M M Nash. It may be freely copied, printed and used providing the authors are credited when appropriate. The conditions of use and ways to acquire the book are given below. JN 941011 / Revised 971130.

Scientific Computing with PCs

John C. Nash
Faculty of Administration
University of Ottawa

Mary M. Nash
Nash Information Services Inc.

ISBN: 0 88769 008 4

Copyright © 1984, 1993 J C Nash and M M Nash

This book is available from Nash Information Services Inc. by electronic mail or pre-arranged FTP transfer. This file, SCPCDOC.PDF, contains the Preface, Contents and an example chapter that you may freely print and use providing the authors are credited.

Cheques for the full document, also available as an Adobe Acrobat (PDF) file, should be for \$10 US or equivalent (plus applicable taxes for Canadian residents) payable to

Nash Information Services Inc.
1975 Bel-Air Drive, Ottawa, Ontario, K2C 0X1, Canada
Fax: (613) 225 6553

Email contacts: jcnash@uottawa.ca, mnash@nis.synapse.net
Web site: <http://www.nashinfo.com/~nis/>

This file formatted on 30 November 1997

Preface

This book tells "what to", "why to" and "when to" use personal computers (PCs) for carrying out scientific computing tasks. There are many books that purport to tell users how to operate particular software packages or to set up and maintain their personal computer hardware. These "how to" volumes have their proper role and utility. Our goal is different: we aim to present ideas about the management decisions involved in using PCs for scientific and technical computing.

The audience for this book is any scientist, engineer, statistician, economist, or similar worker who is considering using a personal computer for all or part of his or her technical work. The ideas we present can be used at the individual or group level. Being ideas about the management of resources, they are kept simple to the extent that readers will think "I know that." We stress, however, that it is the day-to-day practice of management ideas that counts; theoretical knowledge is like a smoke alarm without a battery.

There are three themes to this book:

- Deciding what is or is not reasonable to attempt in the way of scientific computing with a PC;
- Learning about and finding software to carry out our tasks with a minimum of fuss;
- Choosing the tools to match the task to keep costs low.

By PC we will mean any computer system that is managed, used and maintained by a single person. The typical examples are the relatives and descendants of the original IBM PC and the Apple Macintosh, but other machines and workstations have similar properties. Workstation networks and mainframes with system support staff have quite different management strategies. Portable computers used with such larger systems are, however, very likely to be used in a way that requires the user to exercise management skills. Our examples use mainly MS-DOS PCs since these are the most common.

The reader will notice some repetition of ideas we feel important; we also have tried to keep sections of the book self-contained.

The distribution mechanism for this book is an experiment in a new form of publishing using the Internet. We did not intend this to be the case. Unfortunately the publisher who commissioned the book as a venture into a new subject area decided to retreat in the face of recessionary pressures after we had submitted a typeset manuscript. Interest from other publishers did not transform itself into a contract. Our response has been to pay to have the work professionally reviewed — as a good publisher should — and to then edit and produce the work in a form that users can print out for their own use.

The current work has been formatted to reduce the number of pages from about 300 to about 200 by choosing margins that use more of the page area on traditional letter-size paper. We believe that A4 paper users will find that pages fit this size also.

Your comments are welcome.

John & Mary Nash, Ottawa, September 1994

Contents

Part I: Panorama	4.8	Input/output problems — bugs and glitches
Chapter 1: Introduction and Purpose — How Scientific Problems are Solved	4.9	Debugging and trouble-shooting
1.1 History and goals	4.10	Precision problems
1.2 Non-goals	4.11	Size problems
1.3 The scientific method and our audience	4.12	Utilities and support
1.4 Computer configurations of interest	4.13	Summary
1.5 Computing styles		
Chapter 2: Data processing capabilities of PCs		Chapter 5: File management
2.1 Non-mathematical data processing related to calculation	5.1	File creation and naming
2.2 Communications — Access to remote data	5.2	Types and distributions of files
2.3 Automated data collection	5.3	Cataloguing and organizing files
2.4 Input, output, edit, and selection	5.4	Backup
2.5 Scientific text processing	5.5	Archiving
2.6 Macro-editors and data converters	5.6	Version control
2.7 Graphics	5.7	Loss versus security
2.8 Report writers		
2.9 Programming and program development		Chapter 6: Programming Aids and Practices
2.10 Administration of research		6.1 Programming tools and environments
		6.2 Cross-reference listing and data dictionary
		6.3 Flowchart or other algorithmic description
		6.4 Structure and modularity
		6.5 Sub-programs — data flow
		6.6 Programmer practices
Chapter 3: Computational capabilities of PCs		Chapter 7: Size Difficulties
3.1 Good news	7.1	Determining memory and data storage limits
3.2 Application problems	7.2	Choice or adjustment of data structures
3.3 Component computational problems	7.3	Usage map
3.4 Component symbolic problems	7.4	Restructuring
3.5 Getting things to work	7.5	Reprogramming
	7.6	Chaining
	7.7	Change of method
	7.8	Data storage mechanisms for special matrices
Part II: Computing environments		
Chapter 4: Will a PC suffice?		Chapter 8: Timing Considerations
4.1 Hardware	8.1	Profile of a program
4.2 Software	8.2	Substitution of key code
4.3 Interfacing, communications and data sharing	8.3	Special hardware
4.4 Peripherals	8.4	Disk access times
4.5 Operational considerations	8.5	Time/accuracy and other trade-off decisions
4.6 Issues to be addressed	8.6	Timing tools
4.7 Programming effectively		
4.8 The software development environment		

Chapter 9: Debugging	Chapter 13: Problem formulation
9.1 Using output wisely	13.1 Importance of correct formulation
9.2 Built-in debugging tools	13.2 Mathematical versus real-world problems
9.3 Listings and listing tools	13.3 Using a standard approach
9.4 Sub-tests	13.4 Help from others
9.5 Full tests — robustness of code to bad input	13.5 Sources of software
9.6 Full tests — sample and test data	13.6 Verifying results
9.7 Resilience to I/O failure and disk problems	13.7 Using known techniques on unknown problems
9.8 Tests of language translators	
Chapter 10: Utilities — a desirable set	Part IV: Examples
10.1 Editors	Chapter 14: The Internal Rate of Return Problem
10.2 File view, move, copy, archive and backup	14.1 Problem statement
10.3 File display, conversion and print	14.2 Formulations
10.4 File comparison	14.3 Methods and Algorithms
10.5 Directories and catalogs	14.4 Programs or Packages
10.6 Fixup programs	14.5 Some solutions
10.7 Sorting	14.6 Assessment
Chapter 11: Looking after the Baby - Hardware and Operating Practice	Chapter 15: Data Fitting and Modelling
11.1 Cleanliness and a decent physical environment	15.1 Problem Statement
11.2 Bad habits — smoke, sticky fingers, children	15.2 Formulations
11.3 Magnetic disks and their care	15.3 Methods and Algorithms
11.4 Tapes, CDs and other storage media	15.4 Programs or Packages
11.5 Power considerations	15.5 Some solutions
11.6 Interference and radio frequency shielding	15.6 Assessment
11.7 Servicing and spare parts	Chapter 16: Trajectories of Spacecraft
11.8 Configuration maintenance	16.1 Problem statement
Part III: The problem and its solution method	16.2 Formulations
Chapter 12: Steps in Problem Solving	16.3 Methods and Algorithms
12.1 Problem formulation example	16.4 Programs or Packages
12.2 Algorithm choice	16.5 Some solutions
12.3 Algorithm implementation — programming	16.6 Assessment
12.4 Documentation	Chapter 17: Simulating a Hiring Process
12.5 Testing — data entry	17.1 Problem Statement
12.6 Testing — sub-programs	17.2 Formulations
12.7 Testing — complete programs	17.3 Methods and Algorithms
12.8 Production runs	17.4 Programs or Packages
12.9 Modifications and improvements	17.5 Some solutions
	17.6 Assessment

Trademarks

The following table lists, as far as we could determine, the holders of various trademarks mentioned in this work.

Trademark Holder	Trademark(s) or Trade Name(s)
Adobe Systems Inc.	PostScript
American Mathematical Society	T _E X
Apple Computer Corporation	Apple, Macintosh
Aptech Systems Inc.	GAUSS
Borland International, Inc.	BRIEF, dBase III, dBase IV, PARADOX, Turbo Basic, Turbo Pascal, Turbo C, Borland C, Turbo C++, Borland C++, Quattro, Quattro Pro
C Ware Corporation	SEE, DeSmet C, DeSmet ASM88
Conceptual Software Inc.	DBMSCOPY
Corel Corporation	CorelDRAW, CorelPHOTO-PAINT
Crescent Software Inc.	GraphPak
Data Description Inc.	Data Desk
Data Viz Inc	MacLink
Gazelle Systems	OPTune
Gibson Research Inc.	SpinRite
Hewlett-Packard	Hewlett-Packard
Intel Corporation	Intel, 386
International Business Machines Inc.	IBM, RS6000, OS/2
Lotus Development Corp.	Lotus, 1-2-3
The MathWorks Inc.	MATLAB
Microsoft Corporation	Excel, Microsoft, Quick C, MS-DOS, Windows, Quick BASIC, Visual BASIC
Minitab Inc.	MINITAB
Motorola	Motorola, 6800, 68000
Nash Information Services Inc.	SnoopGuard
NeXT Computer Inc.	NeXT
Numerical Algorithms Group Inc.	NAG
Scientific Endeavors Corp.	GraphiC
Stata Corporation	Stata
Sun Microsystems, Inc.	SUN
Soft Warehouse Inc.	DERIVE, muMATH
Strategy Plus Inc.	EXECUSTAT
Stac Electronics Inc.	STACKER
SYSTAT Inc. (now part of SPSS Inc.)	SYSTAT, MYSTAT
Travelling Software Inc.	Laplink
True BASIC Inc.	<i>True BASIC</i>
UNIX System Laboratories Inc.	UNIX
Waterloo Maple Software	Maple
Wolfram Research	Mathematica
WordPerfect Corporation	WordPerfect
Zilog Corporation	Z80

Part IV: Examples



The next four chapters present examples to illustrate the ideas we have been developing so far. We follow these examples with a timing study showing how minor differences in program code may alter running time and other program characteristics, underlining our recommendation to time and test programs. We conclude our illustrations with a case study in the use of graphical tools for analyzing program performance. This achieves two goals — we illustrate how to analyze performance while demonstrating some graphical tools to assist in the analysis.

Chapter 14

The Internal Rate of Return Problem

- 14.1 Problem statement
- 14.2 Formulations
- 14.3 Methods and Algorithms
- 14.4 Programs or Packages
- 14.5 Some solutions
- 14.6 Assessment

This chapter presents the internal rate of return (IRR) problem as a case study of problem formulation and solution.

14.1 Problem Statement

A common calculation to help measure the value of investments is the **internal rate of return**. While not directly a scientific computation, it illustrates many characteristics of computational problems but has easily understood origins. We invest various amounts $I(t)$ in different time periods $t = 0, 1, 2, \dots, n$ and these investments (or costs) return revenues $R(t)$ in the corresponding time periods. The net revenue in each period is then

$$(14.1.1) \quad N(t) = R(t) - I(t) \quad \text{for } t = 0, 1, 2, \dots, n.$$

The "zero'th" period allows for initial investments. The net value of the investment is the sum of the net revenues, but to take account of the time value of money, we discount each net revenue appropriately. The later the net revenue is received, the less it is worth. Given a discount rate r , we create Table 14.1.1.

Table 14.1.1 Schematic layout of costs and revenues for the internal rate of return problem.

Period	Investment	Revenue	Net Revenue	Discounted
0	$I(0)$	$R(0)$	$N(0)=R(0)-I(0)$	$N(0)$
1	$I(1)$	$R(1)$	$N(1)=R(1)-I(1)$	$N(1)/(1+r)$
2	$I(2)$	$R(2)$	$N(2)=R(2)-I(2)$	$N(2)/(1+r)^2$
t	$I(t)$	$R(t)$	$N(t)=R(t)-I(t)$	$N(t)/(1+r)^{(t-1)}$

The sum of the discounted net revenues, or cash flows, is called the **net present value** (NPV) of the investment at the beginning of the investment sequence. The IRR is the value of r such that this NPV is zero. This rate can be compared with interest rates offered by other investments. Uncertainties imply that high precision is not needed in our calculation.

14.2 Formulations

The mathematical formulation of the IRR problem is:

Find the discount rate, r , such that the NPV of the revenue stream $N(t)$, $t=0, 1, 2, \dots, n$ is zero.

$$(14.2.1) \quad NPV(r) = 0 = \sum_{t=0}^n (R(t)-I(t)) / (1+r)^t$$

Letting $1/(1+r) = x$, we have

$$(14.2.2) \quad \sum_{t=0}^n (R(t)-I(t)) x^t = \sum_{t=0}^n N(t) x^t = 0$$

This is the polynomial root finding problem, since the last summation simply defines a polynomial. A polynomial with n coefficients $N(t)$ is said to be of degree $(n-1)$ and has $(n-1)$ roots. Not all the roots of the polynomial need to be real; complex-conjugate pairs are possible. However, the answers for the rate of return must be real. We don't have imaginary deposit rates! It is useful in some approaches to multiply (14.1.3) by $(1+r)^n$,

$$(14.2.3) \quad \sum_{t=0}^n N(t) (1+r)^{(n-t)} = 0$$

The problem can be simplified if we look for just one root between reasonable limits. Clearly a NPV that is negative when $r=0$ means that the investment is a poor one. It is losing money even when there is no "interest" on the investment. So $r=0$ may be chosen as a lower limit to the root. A 50% return per annum ($r=0.5$) is generally considered extremely good, so would be a **reasonable** upper limit. If the return is less than 50%, then $NPV(0.5)$ should be negative — revenues late in the sequence of cash flows don't count as much. However, it is possible that there are multiple roots that arise to confuse the whole situation.

It is wise to calculate and plot the NPV for several likely values of r . This gives us a picture of the NPV at various discount rates and errors in a purely automatic solution. Such graphical checks of calculations should always be performed if possible.

We can perform tests other than a plot of the function. A positive return on the investment, means $NPV(0) > 0$, as mentioned. If we are to have just one root, NPV to decline with increasing r , so the first derivative of $NPV(r)$ with respect to r should be negative at $r=0$. This is easily calculated. We use the chain rule to find:

$$(14.2.4) \quad d NPV(r) / dr = - \sum_{t=1}^n N(t) t / (1+r)^{(t+1)}$$

For illustrative purposes, we will use two sets of cash flows. Figure 14.2.1a presents a straightforward problem; data in Figure 14.2.1b includes some of the potentially tricky features mentioned above.

As suggested in Chapters 12 and 13, we have formulated our problem in two main ways, and have listed some side conditions on the problem and some reality checks for suggested solutions. Our two formulations are:

- Find all the roots of a polynomial and select the one that is appropriate;
- Search for just one root in a restricted range with tests on the applicability of the search.

Figure 14.2.1 Two cash flow streams for the IRR problem;

a. a straightforward problem,

b. a problem with multiple roots.

Problem IRR2A -- a straightforward problem with a simple root (discount rate)				IRR3 -- problem with multiple roots	
Period	Invest	Revenue	Cash Flow	Period	Cash Flow
0	600	0	-600	0	-64000
1	0	300	300	1	275200
2	0	250	250	2	-299520
3	0	600	600	3	-189696
4	2000	0	-2000	4	419136
5	0	0	0	5	41376
6	0	455	455	6	-218528
7	0	666	666	7	-23056

14.3 Methods and algorithms

The main formulations suggest that we need either

- A method for finding all the roots (or zeros) of a polynomial, or
- A general root-finding technique for one root in a specified interval.

Both methods can be found in the literature. Many methods exist for the polynomial roots problem. See Ralston, 1965, Chapter 8 for a historical overview. Unfortunately, most methods have weaknesses, and few will compute all the roots. The method of Jenkins and Traub (Jenkins, 1975) is considered by many workers to be definitive. An alternative approach is to compute the eigenvalues of the companion matrix of the polynomial (Mathworks, 1992, p. 394ff). A general method for solving a single nonlinear equation (one real root of a function) is easier to find (Kahaner, Moler and Nash S G, 1989, Chapter 7; Nash J C, 1990d, Algorithm 18). An advantage is that we can work entirely in real arithmetic.

14.4 Programs or Packages

We now need to decide how to implement our solution method. If we decide to compute all the roots of the polynomial, we need a tool that can compute these. Since the Jenkins (1975) program is part of the ACM Transactions on Mathematical Software collection available through NETLIB we could get this by

electronic mail or a file transfer method. We would then have to write a driver program, compile it and supply the polynomial coefficients.

MATLAB uses the companion matrix approach and has a function `roots()` built in. A useful adjunct to this is the `poly()` function that lets us rebuild the polynomial from its roots as a check on the computations. We note that *DERIVE* will *symbolically* solve for real roots of some nonlinear equations. It is designed to find exact or symbolic solutions, so we cannot be certain it will be of use. There are many choices of general rootfinder software. In addition to the references above, we can use codes from Press et al. (1986) and subsequent volumes. There are also rootfinders in a number of public software collections e.g., NETLIB.

Using spreadsheets we can easily plot $NPV(r)$ for a reasonable range of discount rates r . Some spreadsheets now offer to find roots of functions (solve equations), though student versions on our shelf of Lotus, Quattro and Excel did not have this capability. They have a built-in IRR and NPV functions that we do not think are worth using, as will become clear. It is possible to "program" in the macro language of spreadsheets, and root finding could be carried out this way.

Lotus allows other software to be attached to it. Of these "add-in" programs, What-If Solver (Nash J C, 1991b) provides a capability to find roots of general nonlinear equations, so could be applied here.

14.5 Some Solutions

Our software choice for solving rate-of-return problems would be a straightforward spreadsheet package *without* special additions to compute roots of equations. That is, we forgo the mathematical satisfaction of finding the complete set of roots of the polynomial equation (14.2.2). Because the rate of return is used as a guide to investment management, it is not important to find a very accurate approximation to any single root. Instead, we suggest plotting the NPV against discount rate r and observing where zeros are located. It is easy to add to the table used for plotting and verify NPV for good "guesses" of r . However, we suggest that IRR and NPV functions built into many spreadsheets *not* be used for reasons we give below.

Figure 14.5.1 shows the data and expressions for the straightforward problem defined in Figure 14.2.1a. Figure 14.5.2 shows the graph of $NPV(r)$ versus r , the discount rate. Calculations were carried out with Lotus 1-2-3; Quattro Pro gives equivalent results. Readers should note that the `@NPV()` function within spreadsheets does *not* give us $NPV(r)$ as defined in Equation 14.2.1 without some modifications. If in the spreadsheet our net revenues $N(1) \dots N(9)$ are stored in cells C11 to C19, with the initial net revenue (i.e., the investment) $N(0)$ in C10, the correct expression for the NPV for time $t=0$ at a rate stored in cell B10 is

$$+C10+@NPV(B10,C11..C19)$$

Since the IRR is a built-in function of many spreadsheet packages, it is tempting to use this function. In Figure 14.5.1 this works correctly. DO NOT trust such functions. They will not always work for problems with multiple roots, such as that defined by the net revenues in Figure 14.2.1b. From the graph in Figure 14.5.3 we see that there are solutions near 10%, 50% and -150% and these are confirmed by the worksheet. The `@IRR` function determines this latter root at -143.857% from starting guesses at 0, 0.5, and -1.5, even though the correct value is very close to -143%. It fails (gives ERR as the result) from several other trial starting values. In an attempt to overcome the difficulty, we changed the sign of all net revenues. This should not change the position of the root, but it caused all our attempted `@IRR` calculations to give an ERR result. The NPV at -143.857 as calculated by the `@NPV` function and by direct spreadsheet statements are different. This is probably because the elements of the sum are quite large and digit cancellation is occurring. Note `@IRR` does not find the root at 10% that is likely to be of interest.

We can also program our own solution by rootfinding. In 1980 we built a rate-of-return program in BASIC using character graphics to plot NPV versus r and the rootfinder of Algorithm 18 in Nash J C (1990d).

Figure 14.5.1 Data and expressions for a straightforward internal rate of return problem using a Lotus 1-2-3 spreadsheet.

```
IRR2a.WK1 -- Internal rate of return = 7.827 %

Time period:      0    1    2    3    4    5    6    7    8
Investments -->  600    0    0    0 2000    0
Revenues   -->    0   300  250  600    0    0  455  666  777
=====
Combined    --> -600  300  250  600 -2000    0  455  666  777
Contribution to -600  278  215  479 -1479    0  289  393  425
NPV at r= 7.8275 == Summed NPV contributions = 0.002

NPV (r from @IRR = 7.8275) --> -2E-14
== check on calculated IRR using = -600 + NPV(t=1..t=8)
```

We would not recommend programming this problem now unless it is to be embedded inside some heavily-used investment analysis package.

Using the "all roots" formulation, we can quite in quite straightforward fashion acquire and install the Jenkins (1975) program. This took one of us about an hour to do, including getting the code from NETLIB by electronic mail, downloading it by telephone to our PC, writing and testing a driver program, and then running our problem. A nuisance was finding the right values to use for the radix, number of digits, smallest and largest real numbers that the program needs. These are particular to compiler and computer used; we eventually spent over half a day investigating the possibilities, as illustrated in Figure 14.5.4.

A further difficulty comes in deciding whether roots are admissible. The `roots()` function of MATLAB gives $(1+r)$ values at 1.10, 1.50 and -0.5 having small non-zero imaginary parts. Initial results from the Jenkins program did not seem to match. Using a smaller "machine precision" value, though the arithmetic is unchanged, gave results in closer agreement to MATLAB. Using MATLAB, we reconstruct the polynomial, scale it appropriately, and find the deviation from the original sequence of cash flows. Such an exercise showed that the roots computed by MATLAB indeed gave us back the original problem. We tried the same reconstruction with results from the Jenkins code, varying the tolerance used to judge "small" numbers. In the case of the simple-root problem (data of Figure 14.2.1a) the deviations decreased as the tolerance was decreased. For the multiple-root problem, however, Figure 14.5.4 shows that the solutions vary in quality as the tolerance for zero is made smaller. For both problems, a very small tolerance caused a failure flag to be set. The behaviour of results using three different compilers (Lahey F77L 5.01a, Microsoft Professional Fortran 5, Watcom Fortran 9) were similar but far from identical.

Figure 14.5.5a shows the very simple steps to use the SOLVE capability of the symbolic mathematics package *DERIVE*. We get just three reported roots (*DERIVE* makes no attempt to assess multiplicity). In this case we can SOLVE. This did not give a solution for the more straightforward problem of Figure 14.2.1a, though we note that a good approximation to roots can be found by plotting $NPV(r)$, since *DERIVE*'s graphs have a user-controllable cross-hair for which the position is displayed as well as a ZOOM facility that lets us locate a root as accurately as we like. Alternatively, as shown in Figure 14.5.5b, we can set up a Newton iteration in *DERIVE*, but must remember to set the arithmetic as approximate to get numerical answers. If we leave arithmetic as exact, the iteration fills memory as it takes a great deal of time to "compute" roots.

Figure 14.5.2 Graphical solution of the problem of Figure 14.5.1.

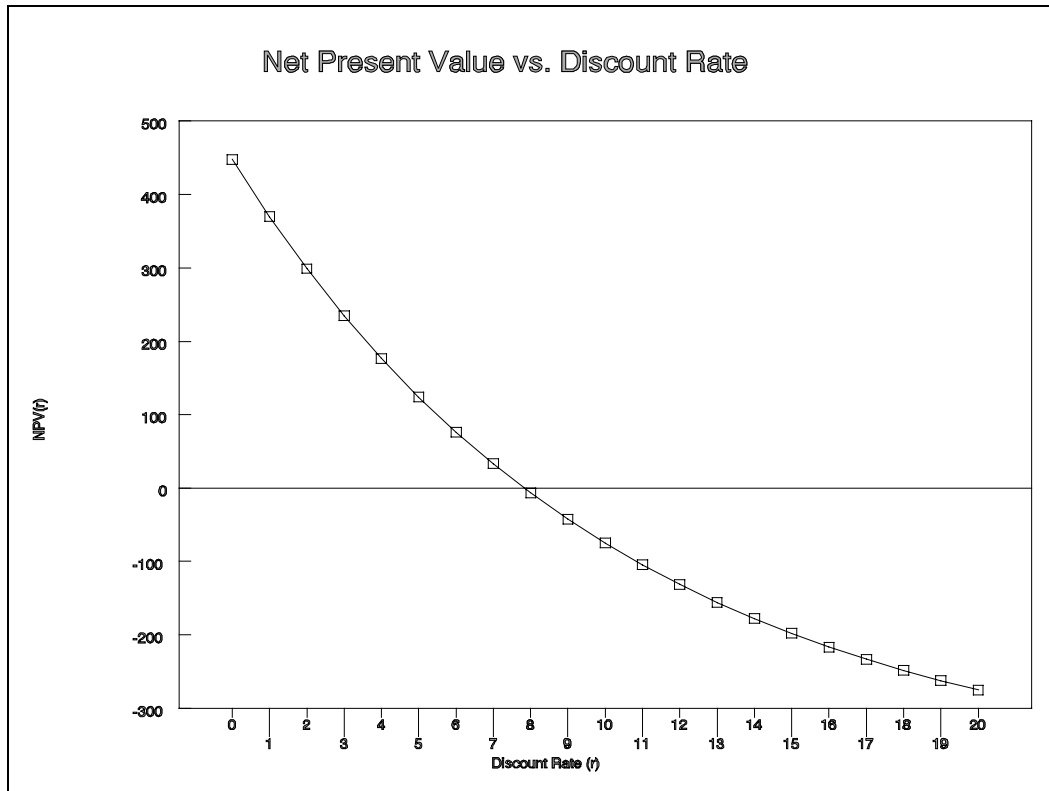


Figure 14.5.3 Graphical solution of the problem of Figure 14.2.1b. The graph is truncated when off-scale.

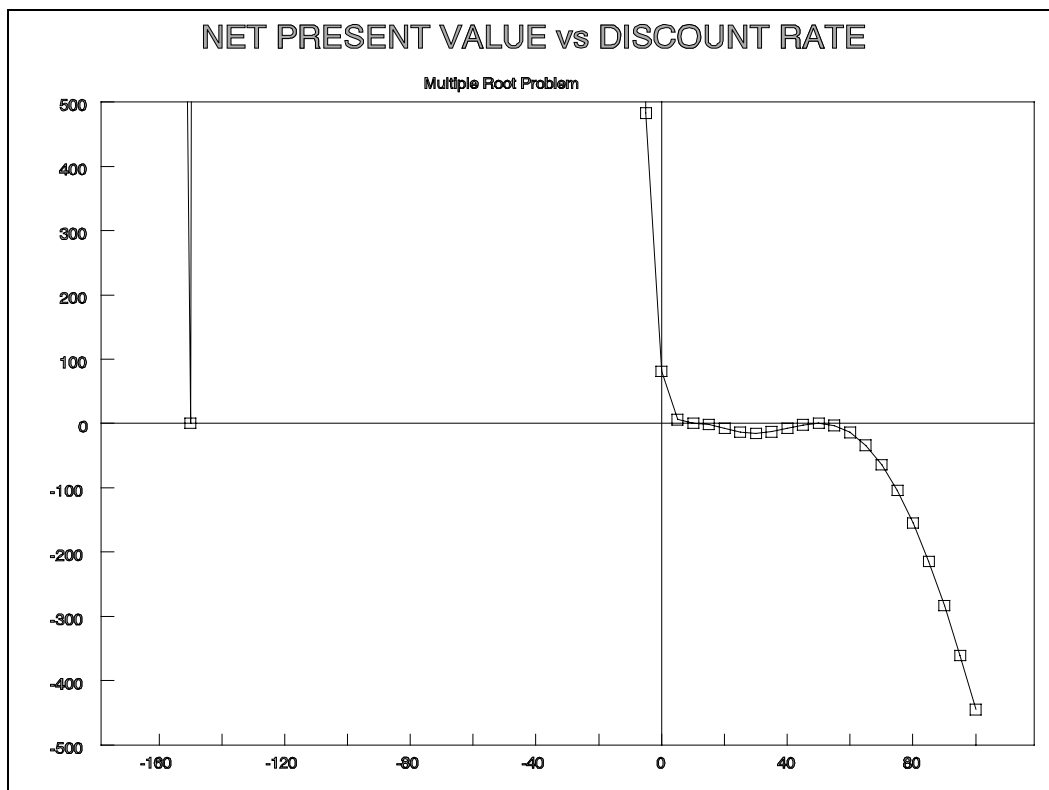


Figure 14.5.4 Graph of the log of the sum of squares of the deviations between the cash flow coefficients and their reconstructions from the computed polynomial roots for the problem described in Figure 14.2.1b.

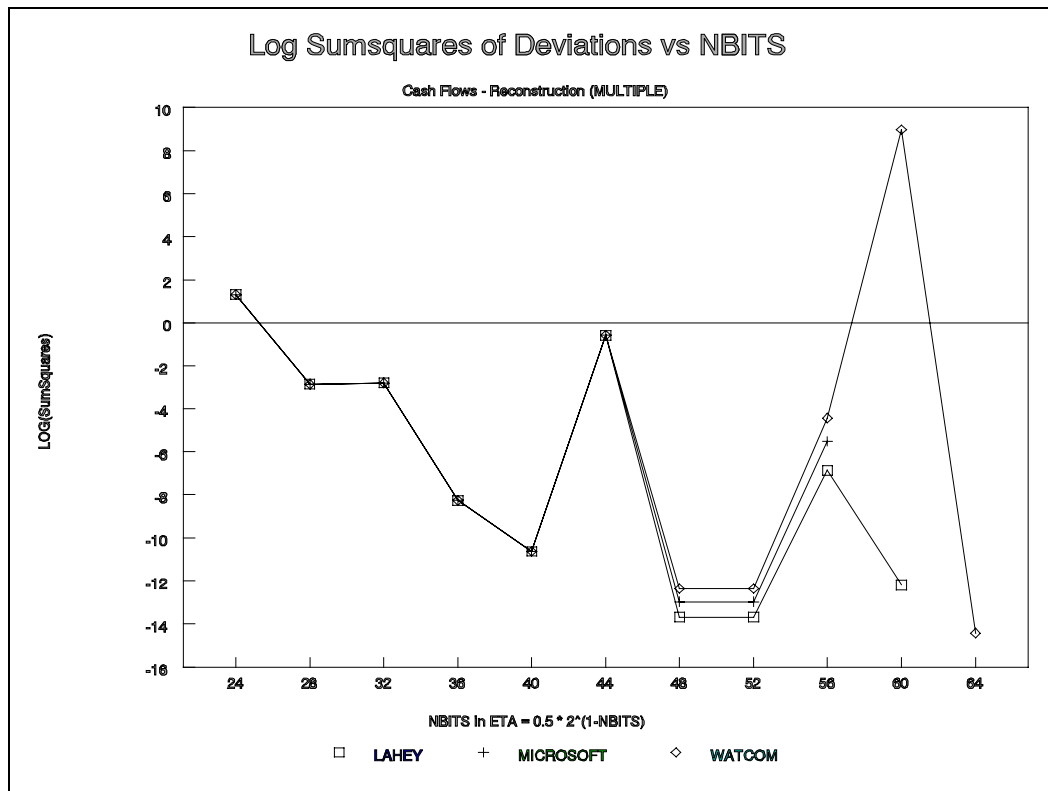


Figure 14.5.5 a) Solution of the internal rate of return problem 14.2.1b by the SOLVE function of *DERIVE*.

$c := [-64000, 275200, -299520, -189696, 419136, 41376, -218528, -23056, 47190, 11979]$

$ply := \text{SUM}(\text{ELEMENT}(c, i) * (1+r)^{(1-i)}, i, 1, 10)$

$(275200 * r^8 + 1902080 * r^7 + 5419264 * r^6 + 8402240 * r^5 + 8072416 * r^4 + 5272416 * r^3 + 2331216 * r^2 + 573462 * r + 64081) / (r+1)^9 - 64000$

$r = 1/10$

$r = 1/2$

$r = -3/2$

Figure 14.5.5 b) Solution of the internal rate of return problem 14.2.1a by a Newton iteration defined in *DERIVE*. Approximate arithmetic is used.

$c := [-600, 300, 250, 600, -2000, 0, 455, 666, 777]$

$ply := \text{SUM}(\text{ELEMENT}(c, i) * (1+r)^{(1-i)}, i, 1, 9)$

$\text{NEWTON}(u, x, x0, n) := \text{ITERATES}(x - u / \text{DIF}(u, x), x, x0, n)$

$\text{NEWTON}(ply, r, 0)$

$[0, 0.0545808, 0.0757515, 0.078245, 0.0782755, 0.0782755]$

14.6 Assessment

The mathematically obvious solution of the IRR problem in terms of polynomial roots led us to annoying detail both in assessing the validity and admissibility of roots and providing appropriate machine constants for our compiler. If one wants to know all the roots, MATLAB or a similar tool seems a better choice. *DERIVE* seems to win the contest on simplicity, though it will not always compute all the roots.

Of course, when we draw a graph of NPV versus discount rate, our eyes only pick out distinct roots. We can confirm these roots by computing the NPV directly in a spreadsheet or with *DERIVE*. *DERIVE* also has the Newton iteration if we want very precise roots.

If IRR values must be supplied automatically, a rootfinder that computes a single root in a given interval is our choice. We would not recommend programming this in the macro language of a spreadsheet. Having tried this, we believe it is too easy to make small but damaging errors placing data in the spreadsheet so the macros can work on it. Instead, we would choose a FORTRAN version of Nash J C (1990d) Algorithm 18 and graphs drawn with VG (Kahaner and Anderson, 1990) so only a reasonably small driver program must be coded.

In solving the example problems here, we brought to light some deficiencies of built-in functions in spreadsheets. Users should run check calculations of functions in any package — errors can and do creep in, so we must protect ourselves as best we can.