UCID- 30050, Part 2
COMPUTER
DOCUMENTATION

LAWRENCE LIVERMORE LABORATORY
University of California/Livermore, California

CONSTRUCTION OF MATHEMATICAL SOFTWARE PART II SOME EXAMPLES OF MATHEMATICAL SOFTWARE

R.E. von Holdt
R.P. Dickinson, Jr.
R.L. Pexton

December 5, 1972

UCID-30050, pt. 2, c.1

32

Prepared for U. S. Atomic Energy Commission under contract no. W-7405-Eng-48

Foreword

This is Part II of the five-part report Construction of Mathematical Soft-ware. Parts I, III, and V were issued previously; Part IV will appear later.

The outline for the complete report is given below; see the following page for a more detailed description of the contents of Part II.

Contents of Part II

CONSTRUCTION OF MATHEMATICAL SOFTWARE
PART II
SOME EXAMPLES OF MATHEMATICAL SOFTWARE

Abstract

This is Part II of a five-part report on the construction of mathematical software. Chapter 1 discusses software for the elementary functions, while Chapter 2 is concerned with input/output number conversion. Chapter 3 is a brief description of EISPACK, a package of subroutines to solve the algebraic eigenvalue problem. Software for calculating Padé approximants, based on the work of I.M. Longman, is described in Chapter 4.

Chapter 1. Software for the Elementary Functions

R.E. von Holdt

1. Introduction

This chapter is concerned primarily with the elementary function routines that are being constructed for the CDC STAR-100 library. The basic techniques are the same as those used in the corresponding routines in the CDC 6600 and 7600 libraries.

1.1. Basic Functions.

The basic functions of the calculus can be described as either primitive or elementary, according to the way in which they are implemented by hardware or software. Thus, the square root function is elementary on the 7600, but primitive on STAR. For example, eleven familiar functions can be placed on a scale of complexity as follows:

PRIMITIVE		ELEMENTARY
+, -, *, /,	$\sqrt{x}$	$\ell n x$, e^x, cos x, sin x, Arctan x, R2P[†]

We should also be able to include number base conversion[**] in the primitive category, since this operation should be done by the hardware. However, while there are register-mode conversion instructions on STAR, there are no stream mode equivalents. We shall consider only the last six (elementary) functions in this chapter.

[†]Rectangular to polar coordinate transformation. (Replaces standard FORTRAN function ATAN2.)

[**]See Chapter 2.

1.2. Machine Dependence. The trends in the use of machine-dependent vs machine-independent programming can be represented as a swinging pendulum. The first computers required machine-dependent coding. As higher level languages (FORTRAN, Algol) became popular, the swing was to all machine-independent programs.

With the advent of more complicated computer architecture (Illiac-IV, STAR), the pendulum has started to swing back the other way. It is fairly clear that one cannot have machine-independent software that covers the entire spectrum without seriously degrading the performance of the more advanced machines.

1.3. Computer Complexity. Computers can be classified by degree of complexity (Table 1.1). To illustrate one of the differences between the 7600 and the 6600, consider the addition of two floating point numbers. The addition process can be broken down into four stages:

1. Compute the exponent difference and determine its sign.
2. Align the smaller operand.
3. Perform the fixed point addition of the fractional parts.
4. Correct for overflow if the number of digits has increased.

On the 6600 the complete process is done by a single add unit and requires four cycles. On the 7600 there are four chained units, one for each stage of the operation. Each unit requires one cycle. Thus, one can start additions on successive cycles, instead of having each addition wait until the preceding addition has completed its four stages.

Table 1.1. Computer complexity.

Degree of complexity	Type of hardware	Example	Status
-2	Serial	pre-6600	Gone
-1	Overlapped	CDC 6600	Dying
0	Overlapped + pipeline	CDC 7600	Living
1†	String + pipeline	CDC STAR-100	Hoped for
1†	Parallel	Illiac-IV	?

†These are both given degree 1 because it is not yet clear which is the more advanced concept.

2. Algorithms

The overall structure of our elementary function software is basically machine independent. However, we become machine dependent when implementing efficiently in machine language. Our aims are:

1. maximum accuracy,
2. minimum computing time, and

3. reasonable space requirements.

Given an infinite amount of memory, we can achieve goals 1 and 2 precisely by having a table containing the exact values of the function for all possible arguments. The third aim requires a trade-off. We have chosen 128 words as being a reasonable table size.

All of the algorithms described here have three basic components:

1. Argument reduction. The arguments are reduced to some standard range in order to both limit the number of calculations and to avoid troublesome regions in the function. Machine dependent parts are the word size and the number base.

2. Approximation of the reduced argument.

3. Function expansion. This is the inverse of the argument reduction. We can no longer guarantee the last bit here.

2.1. Argument Reduction.

If f is the reduced argument, we shall frequently use the notation

$$f = H + L,$$

where H is the high-order part (usually, the first seven bits) and L is the low-order part. Argument reduction for typical functions is described below.

2.1.1. $\ell n\ x,\ x > 0$.

$\ell n\ x = \ell n(2^e \cdot f) = e \cdot \ell n\ 2 + \ell n\ f,\ \frac{1}{2} \le f < 1$. This is the "natural" reduction assuming normalized input. Further reduction is needed, because the series doesn't converge fast enough.

$$f = H + L = H(1 + z), \text{ where } z = L\left(\frac{1}{H}\right).$$

$$\ell n\ f = \ell n\ H + \ell n(1 + z).$$

Tables needed:

128 words for $\frac{1}{H}$ (to avoid division)

128 words for $\ell n\ H$

Approximation needed: $\ell n(1 + z)$

Expansion formula: $\ell n\ x = e \cdot \ell n\ 2 + \ell n\ H + \ell n(1 + z)$ (1.1)

2.1.2. e^x, $e^x < \infty$ (largest number in the machine)

$$e^x = 2^{x\cdot\left(\frac{1}{\ell n\ 2}\right)} = 2^{I-f},\ 0.5 \le f < 1.$$

Note that I gives the exponent of the result immediately.

$$2^{-f} = e^{-f \ln 2} = e^{H+L} = e^{H} \cdot e^{L}$$

Table needed:

128 words for e^{H}.

Approximation needed: e^{L}.

Expansion formula: $2^{I} \cdot e^{H} \cdot e^{L}$. (1.2)

2.1.3. Sine and cosine (both in one routine). The argument x is given in radians. We change to revolutions:

$$y = x\left(\frac{1}{2\pi}\right) = I + f.$$

If I is so large that f has 4 or fewer good bits $\left(\frac{1}{16}\text{ revolution}\right)$, the routine tilts; i.e., an indefinite is generated.

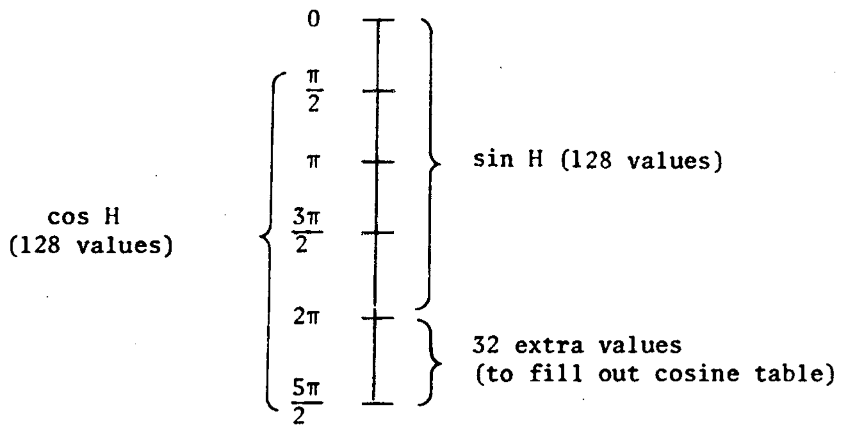
To compute sin f and cos f, where f = H + L, use the identities

$$\sin f = \sin H \cos L + \cos H \sin L \tag{1.3}$$

$$\cos f = \cos H \cos L - \sin H \sin L \tag{1.4}$$

Tables needed:

160 words of overlapped sin H, cos H.



Approximations needed: sin L, cos L (for L in revolutions).

Expansion formulas:

$$\left.\begin{aligned}\sin x &= \sin f\\ \cos x &= \cos f\end{aligned}\right\}\text{ using Eqs. (1.3 and 1.4).}$$

2.1.4. Arctan x.

Arctan x = SIGNF (Arctan |x|, x)

For |x| > 1, Arctan $|x| = \frac{\pi}{2} - \text{Arctan}\left(\frac{1}{|x|}\right)$.

While a test is inevitable, the routine actually forms both $|x|$ and $\frac{1}{|x| + \varepsilon}$ (ε a very small number, to prevent division by zero) and generates a control vector that tells which is smaller.

For $|x| < 1$,

$$|x| = H + L$$

$$\text{Arctan } |x| = \text{Arctan } H + z.$$

Using the trigonometric identity

$$\tan (a - b) = \frac{\tan a - \tan b}{1 + \tan a \tan b},$$

we see that

$$\tan z = \frac{|x| - H}{1 + |x| \cdot H} = \frac{L}{1 + |x| \cdot H}$$

so that

$$z = \text{Arctan} \left(\frac{L}{1 + |x| \cdot H} \right).$$

Table needed:

128 words for Arctan H.

Approximation needed: Arctan $\frac{L}{1 + |x| \cdot H}$

2.2. Approximations. Having reduced the argument to appropriate ranges, we need some economized Maclaurin series approximations for the reduced arguments L, where $|L| < 1/128$. The error curve for the ordinary truncated series is shown in Fig. 1.1a.

One can adjust the curve downward by appropriate modification of the linear coefficient, giving a smaller maximum error (Fig. 1.1b).

Further improvements in the approximation can be made by adjusting other coefficients. The general procedure is as follows:

1. Truncate Maclaurin's series at some point beyond the number of terms you want to use. Carry 2 or 3 extra terms, depending on your estimate for the error in the tail.
2. Expand the resulting polynomial in Chebyshev polynomials.
3. Truncate to the desired degree. The truncation error is bounded by the sum of the absolute values of the coefficients of the omitted terms.
4. Convert back to powers of x.

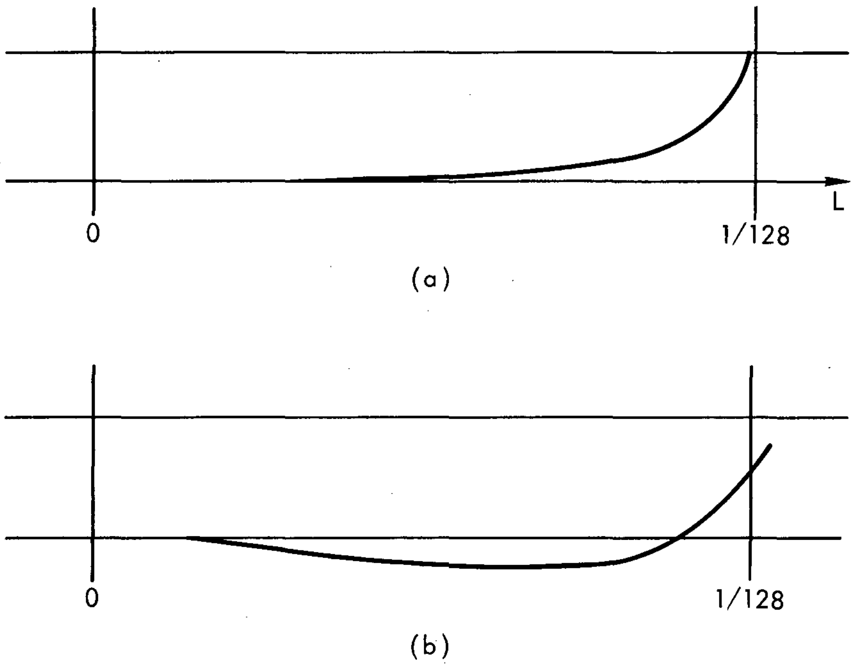


Fig. 1.1. Error curves for the ordinary truncated series (a) and the same series after modification of the linear coefficient (b).

3. Errors

There are two ways in which user errors can be handled.

Garbage in $\longrightarrow$ Garbage out (gigo)

vs

Baby-sitting the user (bstu)

The STAR elementary function routines will use a modified gigo approach; invalid arguments will produce indefinite results. Regarding bstu, the STAR library will allow an optional cut-in. That is, it will be possible to run in "debug mode." In this case all errors will be trapped and many diagnostics provided. For example, the result vector from the logarithm function will be checked for indefinites when the program is in the "debug mode."

Chapter 2. Software for Input/Output Conversion

R. E. von Holdt

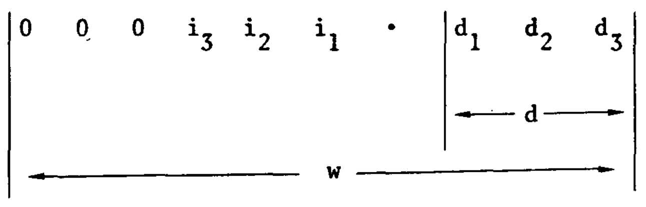
1. Introduction

The original FORTRAN I/O routines used at LLL on the CDC 6600 and 7600 were written in LRLTRAN by John Ranelletti. These notes describe the input/output algorithms as modified by the author.

2. Input

We shall first describe how to handle the conversion of a fixed point (F-type) number from ASCII (decimal) to floating point (binary). It will then be obvious how to handle integer and E-type input. The algorithm has four basic steps:

1. Skip over leading zeros to find first nonzero character.
2. Locate the decimal point, converting the integer part in the process.
3. Convert the fraction.
4. Convert the exponent.



The first step is self-explanatory. In step 2, the accumulated integer I is computed by a so-called "shifty add." The shifty add makes use of the fact that when I*10 is represented as I*8 + I*2 the two multiplications are done by shifts. This process is extremely fast. This operation is also overlapped with a look-ahead to see if the next character is a decimal point (or other field terminator). If so, we end stage 2 with I as the completely converted integer part. If the next character is a digit i_k, we compute $I = I*10 + i_k$ by a simple add and repeat the process.

We summarize the above process in the example given above, where I has three digits:

$I \leftarrow i_3$	
Compute $I*10 = i_3*10$ (shifty add)	Is i_2 a decimal point? No
$I \leftarrow I*10 + i_2 = i_3*10 + i_2$	
Compute $I*10 = i_3*100 + i_2+10$ (shifty add)	Is i_1 a decimal point? No
$I \leftarrow I*10 + i_1 = i_3*100 + i_2*10 + i_1$	
Compute I*10 (shifty add)	Is • a decimal point? Yes
Terminate with current I	

Note that we have unnecessarily computed I*10 at the last step, but this calculation is overlapped with the look-ahead and costs nothing.

The correct way to perform stage 3 of the conversion is to continue converting $i_3\ i_2\ i_1\ d_1\ d_2\ d_3$ to binary, as an integer, keeping track of the number of digits (d) to the right of the decimal point. Multiply by 10^{-d} at the end. This uses a table look up and "multiply round" instruction.

The exponent $(e_1e_2e_3)$ of a floating point number is a binary integer that can be split into two 5-bit parts (convert to base 32). Tables of 32 words each are used to store the multipliers 10^{i_a}, 10^{32i_b} (where $e_1e_2e_3 = 32i_a + i_b$). These are exact for positive exponents, but we can expect some slight inaccuracy for negative exponents. Two tables each for positive and negative exponents are used, for a total of 128 words of table. The conversion is completed when the exponent is added to d and the final "look-up" and multiplication performed. The inaccuracy in the representation of $10^{e_1e_2e_3-d}$ and the resulting multiplication are the only sources of error in this computation. In our example, the final result is

$$\left(i_3*10^5 + i_2*10^4 + i_1*10^3 + d_1*10^2 + d_2*10 + d_3\right)*10^{e_1e_2e_3-3}.$$

An alternative way to do stage 3, and the method that was used in the original routine, is to compute the decimal part as

$$d_1*.1 + d_2*.01 + d_3*.001 + \ldots \quad .$$

The disadvantages of this procedure are several.

1. The multipliers .1, .01, .001, ... cannot be exactly represented in a binary machine.
2. The multiplication d_k*10^{-k} is much slower than a shifty add.

3. Because the operations are done sequentially, from left to right, much inaccuracy is introduced into the sum by effective truncation of the smaller terms d_k*10^{-k} $(k > 1)$ to the noise level of d_1*10^{-1}. (This is the most serious defect.)

3. Output

3.1. Floating Point Numbers. The algorithm for converting from floating point (Binary) to ASCII (decimal) is based on the assumption that most floating point numbers lie in the range

$$10^{-15} \leq |x| \leq 10^{15}.$$

It proceeds in three steps:

1. Reduce number to range

$$10^{-15} \sim B \leq |x| < 1.$$

Here B is a "fuzzy" bound which is strictly greater than 10^{-16}.

2. Further reduce to range

$$10^{-1} \leq |x| \leq 1.$$

Thus $|x|$ is now a pure fraction with nonzero leading decimal digit.

3. Convert fraction.

From this point on we shall use x to denote $|x|$.

The first range reduction (step 1) is shown schematically in Fig. 2.1. Note that little or no calculation is done in this step if the number is in the range indicated by the inequality ($10^{-15} \sim B < |x| < 1$). E is used to accumulate the (decimal) exponent.

Figure 2.2 shows the original version of step 3.

We now have $x \cdot 10^E$, where $10^{-1} \leq x < 1$. This reduction suffers from the fact that one can go through the loop from 0 to 15 times. This puts average computer time (assuming prefetching and an average of 6 iterations) at about 60 cycles, with a range of 10 to more than 150 cycles.

Step 2 can be improved by using a loop, where the loop control is the modifier of E, initialized to 8 (Fig. 2.3). It is down-shifted at each step, with termination on zero. The comparands 10^{-2^k} and the multipliers 10^{2^k} are prefeteched from tables according to the following algorithm:

a. Right after each multiply, fetch next multiplier.

b. Right after each subtract $(x - 10^{-2^k})$, and before the test, fetch the next comparand.

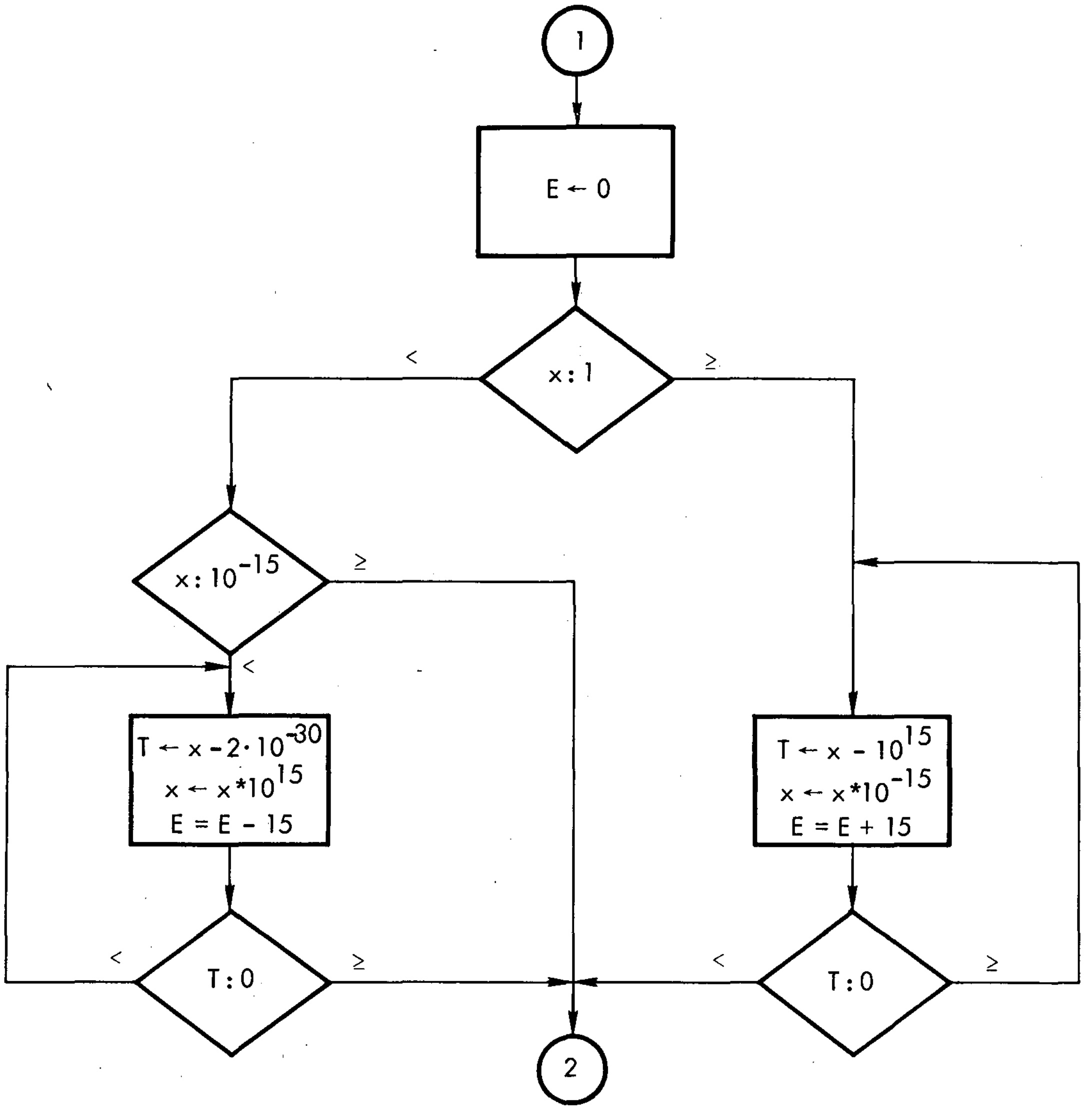

1
$E \leftarrow 0$
$x : 1$
<
≥
$x : 10^{-15}$
≥
<
$T \leftarrow x - 2 \cdot 10^{-30}$
$x \leftarrow x * 10^{15}$
$E = E - 15$
$T \leftarrow x - 10^{15}$
$x \leftarrow x * 10^{-15}$
$E = E + 15$
<
$T : 0$
≥
<
$T : 0$
≥
2

Fig. 2.1. Flow chart showing first range reduction.

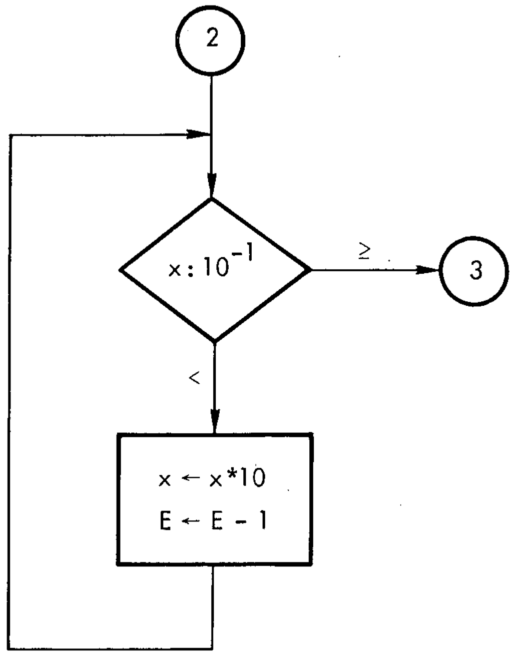


Fig. 2.2. Second range reduction — original version.

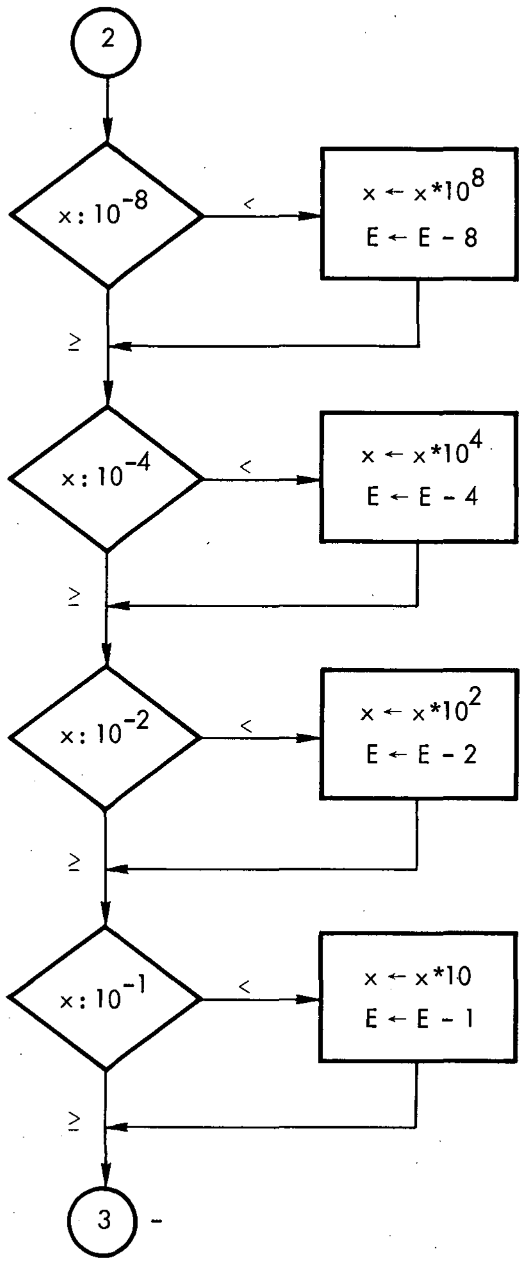


Fig. 2.3. Second range reduction — improved version.

In this way, drop-through costs 10 cycles, while the other branch takes 11 cycles. Thus, the timing is nearly constant, varying between 40 and 44 cycles.

We now have a pure fraction to convert (step 3).

$$f = .f_1 f_2 f_3 \dots$$

We crank out the f_i a digit at a time until the field is filled. We multiply by 10. The integer part of 10f is the first digit, f_1, which is converted to ASCII and added to the number being accumulated. We replace f with $.f_2 f_3 \dots$ and repeat the process.

3.2. Integers. Because the remainder of a division on the 6600 or 7600 is obtainable only by multiplying the quotient by the divisor and subtracting the result from the dividend, the normal digit-at-a-time conversion process for integers is very costly. The solution; convert to a fraction and use the process described above.

Because one can have integers with up to 59 bits, whereas a floating point fraction has 48 bits, it is necessary to do some preprocessing for large integers. If $10^{14} < I \leq 10^{18}$ (has more than 14 digits) we have:

$I_{18}\ I_{17}\ I_{16}\ I_{15}$	$I_{14}\ I_{13} \dots I_2\ I_1$

We divide by 10^{18}, getting the quotient I_{18} a bit at a time and the exact remainder by repeated subtractions. Similarly for I_{17}, I_{16}, I_{15}.

We now assume $0 < I < 10^{14}$. In order to guard against possible bit losses in the conversion process, a fractional part of 1/2 is added before the integer is floated. We now do the reverse of the second reduction process above. (Comparands 10^8, 10^{14}, 10^2; multipliers 10^{-8}, 10^{-4}, 10^{-2}.)

Stopping after the third iteration, we get a number between 1 and 10, as well as a digit count:

$$I \cdot F_1\ F_2\ F_3 \dots$$

We convert this as in step 3, above. Note that by prefetching, the innermost loops go seven times as fast as the FORTRAN version.

4. Vector Output on STAR

We conclude with a proposal for a way to make vector output "sing" on the STAR. The output will be in fixed format, eight vectors across the page, 16 characters per field. For each vector, a scan will determine the maximum (in absolute value) element. If $10^k \leq \max |x| < 10^{k+1}$, the exponent k + 1 will be printed at the top of the column. The elements (adjusted by 10^{k+1}) then follow in F16.0 format, but without the decimal point. (For half words, we will get 16 across the page, with fields 8 characters long.) The choice of fieldwidth limitation is based on the fact that the character vectors will have to be transposed to convert to the necessary line-formatted page for output. The STAR has a hardware instruction for an 8 × 8 matrix transpose.

V_1	V_2	V_3		V_8
$(k_1 + 1)$	$(k_2 + 1)$	$(k_3 + 1)$		$(k_8 + 1)$
$V_1(1)$	$V_2(1)$	$V_3(1)$	...	$V_8(1)$
$V_1(2)$	$V_2(2)$	$V_3(2)$		$V_8(2)$
.	.	.		.
.	.	.		.
.	.	.		.

If the full width is used (14 digits), there will be a sign and a single space separating columns in the full word case. An option to specify the maximum number of significant digits desired could be provided.

Chapter 3. EISPACK: Software for the Algebraic Eigenvalue Problem

R.P. Dickinson, Jr.

Let A be a matrix. Then a vector v is called an eigenvector of A if $v \neq 0$ and if $Av = \lambda v$, where λ is a scalar. The value λ associated with v is called an eigenvalue of A. An important problem of numerical analysis is to find all or some eigenvectors and eigenvalues of a given matrix A (the eigen-system of A). Because of the many different types of matrices, no single algorithm exists to solve this problem. EISPACK is a collection of algorithms designed to solve the algebraic eigenvalue-eigenvector problem.

EISPACK consists of some thirty FORTRAN subroutines or algorithms. A particular matrix problem only demands a few of these routines. Part of the EISPACK procedure is the selection of the correct set of routines needed by the user. This is done at LLL by a question and answer document which leads the user to the correct set.† EISPACK was written at Argonne National Laboratory, under the direction of W. Cowell and C. Moler. The system was designed with the help of J. Wilkinson,[1] a world authority on these problems. Examination of the EISPACK system, including the Argonne backing, will give a good understanding of what good software should be.

†This document will appear separately.

The following material attempts to summarize some of the points which Argonne has made concerning the EISPACK project. Argonne is exploring how scientific computer users can collaboratively test and certify mathematical software. To this end it has chosen two sets of mathematical routines to work on: EISPACK and a special functions package (see Ref. 2).

Argonne also wishes to establish itself as a distribution center for these systems, thus gaining experience in the distribution aspects of mathematical software. The responsibility of such a center would be to guarantee the user of carefully tested work. The user is guaranteed a reliable party to stand behind the package.[3] Hopefully, problems encountered by isolated users will be reported to the central distributor so the experience can be shared. A central distributor has many advantages. It becomes their responsibility to keep abreast of the software state-of-the-art.

With this last thought in mind, Argonne has attempted to write a modular package (thirty subroutines). Any given algorithm might be improved without destroying the flow of the total system. The modularity of EISPACK also gives the user more than one way of solving his problems, depending on his needs. For example, storage might be at a premium. In this case a slower path with less storage might be taken. Also as a distributor, Argonne has written in ANSI standard FORTRAN, a language compatible with most installations. Thus portability also has been incorporated.

Another problem of the distributor is test certification. In this case, Wilkinson has supplied many stringent tests. However, further testing has been made by drawing field test centers into the project. NATS (NSF funded, tests at Argonne, Texas, and Stanford) is part of this effort. The installation at the University of Texas, with investigator Ikebe, is based on CDC 6600. The other installations (Cowell at Argonne and Moler at Stanford) use the IBM 360. Argonne has expanded its field centers to include LLL. Argonne has held workshops in which the field centers have shared their experiences. By observing the Argonne effort many valuable pointers have been gained in the dissemination and creation of good software.

In addition to the basic system EISPACK, a further code, an EISPACK controller, has been written by J. Boyle at Argonne. This system eliminates most of the questions a user must answer before using EISPACK. It automatically choses the routines necessary. In this system the user may treat EISPACK more as a black box. Boyle's system is based on IBM 360. Implementation of Boyle's system is not believed feasible at LLL at this time.

Our plans are to automate the question and answer procedure through teletype control, although documents will be available also. The routines themselves, with drivers or stand-alone, will be stored on Photostore, with backup tape and cards. After preliminary questions by the user on the teletype, the appropriate routines will be loaded from Photostore to the user's disk files. At this point the user might execute the deck or have it punched for later use. We also hope to share the LLL experiences with EISPACK as NATS representatives.

In conclusion, EISPACK is a quality-designed piece of software. It has been designed by experts. It is backed by a prestigious organization. It has been exhaustively field tested. It is highly modular and flexible. It is portable. It is expandable. It is well documented. I appreciate Argonne's model of quality software and its model as a distributor.

Chapter 4. Calculating Padé Approximants

R.L. Pexton

1. Introduction

Power series, continued fractions, or rational fractions are often utilized in the numerical and analytical development of problems in the physical sciences. The ability to accurately and rapidly generate rational fractions from given power series, as claimed by Longman,[4] is therefore, a most welcome addition to our know-how. Our analytic development closely resembles that of Longman, but we believe our subroutine to be superior to his.

2. Theory

Given a power series, $f(x) = \sum_{n=0}^{\infty} C_n x^n$, we desire to calculate coefficients for rational approximants to f(x). Let

$$U_{\mu\nu} = \sum_{i=0}^{\nu} \alpha_i x^i$$

$$V_{\mu\nu} = \sum_{j=0}^{\mu} \beta_j x^j$$

$$E_{\mu\nu} = \frac{U_{\mu\nu}}{V_{\mu\nu}} .$$

The elements $E_{\mu\nu}$ can be displayed in a Padé table:

$$\begin{array}{cccccc}
E_{00} & E_{01} & E_{02} & E_{03} & \cdots & E_{0n} \\
E_{10} & E_{11} & E_{12} & E_{13} & \cdots & E_{1n} \\
E_{20} & E_{21} & E_{22} & E_{23} & \cdots & E_{2n} \\
\vdots & & & & & \\
E_{n0} & E_{n1} & E_{n2} & E_{n3} & \cdots & E_{nn}
\end{array} .$$

For our rational approximants we require that

$$\sum_{n=0}^{\infty} C_n x^n - \frac{U_{\mu\nu}}{V_{\mu\nu}} = 0(x^{\mu+\nu+1})$$

i.e.,

$$\left(C_0 + C_1x + C_2x^2 + \ldots\right)\left(\beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_\mu x^\mu\right)$$

$$= \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_\nu x^\nu + 0(x^{\mu+\nu+1}) \qquad (4.1)$$

Note that the C_i are given and that some powers of x may be multiplied by zero-valued coefficients.

Equating like powers of x produces $\mu + \nu + 1$ equations in $\mu + \nu + 2$ unknowns. This one degree of freedom permits us to assign $\beta_0 \equiv 1$. Classically the remaining α's and β's have been evaluated by means of determinants. One could also use matrix inversion techniques.

Example: For $\mu = \nu = 2$

$$\begin{array}{llllll}
C_0\beta_1 + & & - & \alpha_1 + & & = -C_1 \\
C_1\beta_1 + & C_0\beta_2 - & & - & \alpha_2 & = -C_2 \\
C_2\beta_1 + & C_1\beta_2 + & & & & = -C_3 \\
C_3\beta_1 + & C_2\beta_2 + & & & & = -C_4 \quad .
\end{array}$$

In matrix notation

$$\begin{pmatrix} C_0 & 0 & -1 & 0 \\ C_1 & C_0 & 0 & -1 \\ C_2 & C_1 & 0 & 0 \\ C_3 & C_2 & 0 & 0 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} -C_1 \\ -C_2 \\ -C_3 \\ -C_4 \end{pmatrix} .$$

Symbolically, $Ax = y$ and $x = A^{-1}y$.

These results can more readily be evaluated by Longman's elegant recursive process.

In detail, equating similar powers of x in Eq. (4.1) produces

$$\left.\begin{array}{l} C_0\beta_0 = \alpha_0 \\ C_1\beta_0 + C_0\beta_1 = \alpha_1 \\ C_2\beta_0 + C_1\beta_1 + C_0\beta_2 = \alpha_2 \\ \cdot \\ \cdot \\ \cdot \\ C_\nu\beta_0 + C_{\nu-1}\beta_1 + C_{\nu-2}\beta_2 + \ldots + C_{\nu-\mu}\beta_\mu = \alpha_\nu \end{array}\right\} \nu + 1 \text{ equations}$$

$$\left.\begin{array}{l} C_{\nu+1}\beta_0 + C_\nu\beta_1 + \ldots + C_{\nu-\mu+1}\beta_\mu = 0 \\ \cdot \\ \cdot \\ \cdot \\ C_{\nu+\mu}\beta_0 + C_{\nu+\mu-1}\beta_1 + \ldots + C_\nu\beta_\mu = 0 \end{array}\right\} \mu \text{ equations.}$$

Note that for negative subscripts $C_p = 0$.

The first row of the Padé table, $E_{0\nu}$, can be written down at once.

$$E_{00} = C_0$$

$$E_{01} = C_0 + C_1x$$

$$E_{02} = C_0 + C_1x + C_2x^2$$

$$\cdot$$

$$\cdot$$

$$\cdot$$

$$E_{0n} = C_0 + C_1x + C_2x^2 + \ldots + C_nx^n.$$

Now examine the elements, $E_{\nu 0}$, of the first column of the Padé table. From

$$C_0\beta_0 = \alpha_0$$

$$C_0\beta_1 + C_1\beta_0 = 0$$

$$C_0\beta_2 + C_1\beta_1 + C_2\beta_0 = 0$$

$$\cdot$$

$$\cdot$$

$$\cdot$$

$$C_0\beta_\mu + C_1\beta_{\mu-1} + \ldots + C_\mu\beta_0 = 0$$

we may readily solve for the β's, since $\beta_0 \equiv 1$, e.g.,

$$C_0 = \alpha_0$$

$$\beta_1 = -C_1/C_0$$

$$\beta_2 = -(\beta_1 C_1 + C_2)/C_0$$

$$\beta_3 = -(\beta_2 C_1 + \beta_1 C_2 + C_3)/C_0$$

etc.

The elements of the first column will be

$$C_0$$

$$\frac{C_0}{1 + \beta_1 x} = \frac{C_0}{1 - C_1/C_0 x}$$

$$\frac{C_0}{1 + \beta_1 x + \beta_2 x^2} = \frac{C_0}{1 - C_1/C_0 x - \frac{C_1\beta_1 + C_2}{C_0} x^2}$$

etc.

Longman's recursion formulae interrelate the following elements of the Padé table:

$$\begin{array}{ll} E_{\mu-1,\nu-1} & E_{\mu-1,\nu} \\ E_{\mu,\nu-1} & \end{array}$$

Knowing $E_{\mu-1,\nu-1}$ and $E_{\mu-1,\nu}$ allows Longman to obtain the α's for $E_{\mu,\nu-1}$, and knowing $E_{\mu-1,\nu-1}$ and $E_{\mu,\nu-1}$ allows him to obtain the β's for $E_{\mu-1,\nu}$.
The first row and first column provide the necessary starting conditions.

Demonstration:

For $E_{\mu\nu}$ we have

$$\left(C_0 + C_1 x + C_2 x^2 + \ldots\right)\left(\beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_\mu x^\mu\right)$$
$$= \left(\alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_\nu x^\nu\right) + 0(x^{\mu+\nu+1}).$$

For $E_{\mu,\nu-1}$

$$\left(C_0 + C_1 x + C_2 x^2 + \ldots\right)\left(b_0 + b_1 x + b_2 x^2 \ldots + b_\mu x^\mu\right)$$
$$= \left(a_0 + a_1 x + a_2 x^2 + \ldots + a_{\nu-1} x^{\nu-1}\right) + 0(x^{\mu+\nu}). \quad (4.2)$$

For $E_{\mu-1,\nu}$

$$\left(C_0 + C_1x + C_2x^2 + \dots\right)\left(B_0 + B_1x + B_2x^2 + \dots + B_{\mu-1}x^{\mu-1}\right)$$
$$= \left(A_0 + A_1x + A_2x^2 + \dots + A_\nu x^\nu\right) + 0(x^{\mu+\nu}). \qquad (4.3)$$

Subtracting Eq. (4.2) from Eq. (4.3) gives

$$\left(C_0 + C_1x + C_2x^2 + \dots\right)\Big[(b_0 - B_0) + (b_1 - B_1)x + (b_2 - B_2)x^2$$
$$+ \dots + (b_{\mu-1} - B_{\mu-1})x^{\mu-1} + b_\mu x^\mu\Big]$$
$$= (a_0 - A_0) + (a_1 - A_1)x + \dots + (a_{\nu-1} - A_{\nu-1})x^{\nu-1} - A_\nu x^\nu + 0(x^{\mu+\nu}).$$

Since $b_0 = B_0 = 1$ and $a_0 = A_0 = C_0$, we may divide by x to obtain

$$\left(C_0 + C_1x + C_2x^2 + \dots\right)\Big[(b_1 - B_1) + (b_2 - B_2)x$$
$$+ \dots + (b_{\mu-1} - B_{\mu-1})x^{\mu-2} + b_\mu x^{\mu-1}\Big]$$
$$= (a_1 - A_1) + (a_2 - A_2)x + \dots + (a_{\nu-1} - A_{\nu-1})x^{\nu-2} - A_\nu x^{\nu-1} + 0(x^{\mu+\nu-1}).$$

For a normal Padé table[5] each element is irreducible.[†] Therefore, $b_1 \neq B_1$ and we may divide by $(b_1 - B_1)$ to obtain

$$\left(C_0 + C_1x + C_2x^2 + \dots\right)\left(r_0 + r_1x + r_2x^2 + \dots + r_{\mu-1}x^{\mu-1}\right)$$
$$= \left(s_0 + s_1x + s_2x^2 + \dots + s_{\nu-1}x^{\nu-1}\right) + 0(x^{\mu+\nu-1})$$

which, of course, is $E_{\mu-1,\nu-1}$.

$$R_0 = 1 \qquad\qquad s_0 = \frac{a_1 - A_1}{b_1 - B_1} = C_0$$

$$r_1 = \frac{b_2 - B_2}{b_1 - B_1} \qquad\qquad s_1 = \frac{a_2 - A_2}{b_1 - B_2}$$

[†]Theorem. In a normal Padé table, the approximant occupying the square [p,q] has, in its simplest terms, numerator and denominator whose degrees are exactly q and p, respectively.[6]

$$r_2 = \frac{b_3 - B_3}{b_1 - B_1} \qquad s_2 = \frac{a_3 - A_3}{b_1 - B_1}$$

$$\vdots \qquad \vdots$$

$$r_{\mu-2} = \frac{b_{\mu-1} - B_{\mu-1}}{b_1 - B_1} \qquad s_{\nu-1} = \frac{-A_\nu}{b_1 - B_1}$$

$$r_{\mu-1} = \frac{b_\mu}{b_1 - B_1} \quad .$$

Using the last results, e.g., $\frac{r_{\mu-1}}{b_\mu} = \frac{1}{b_1 - B_1}$ and $\frac{-s_{\nu-1}}{A_\nu} = \frac{1}{b_1 - B_1}$, we solve for B's and a's.

$$B_1 = b_1 - \frac{b_\mu}{r_{\mu-1}} \qquad a_1 = A_1 - \frac{s_0 A_\nu}{s_{\nu-1}}$$

$$B_2 = b_2 - \frac{r_1 b_\mu}{r_{\mu-1}} \qquad a_2 = A_2 - \frac{s_1 A_\nu}{s_{\nu-1}} \;.$$

$$\vdots \qquad \vdots$$

$$B_{\mu-1} = b_{\mu-1} - \frac{r_{\mu-2} b_\mu}{r_{\mu-1}} \qquad a_{\nu-1} = A_{\nu-1} - \frac{s_{\nu-2} A_\nu}{s_{\nu-1}} \;.$$

3. Subroutine Considerations

Longman's published subroutine, PADE, closely follows his analytic development. While this approach provides near crystal clarity, it unfortunately results in a code that is inefficient with respect to execution time and storage requirements. Remarkable improvement in both categories is easily achieved. The improved subroutine is as displayed in Table 4.1. Ultimate efficiency is not claimed. Trivial savings result from a rearrangement of DO loop 7. Note, however, that permuting subscripts I and J permits us to delete loops 11 and 17.

Visualize the AL array as a 3-dimensional parallelepiped (Fig. 4.1).

Let a plane pass through A C D E. The elements of AL will all be stored in the sector A B C D E F. A similar argument holds with respect to the BE array. Hence if we rotate the BE array we may store the nonzero elements in the zero segment of the AL array (slightly modified). One should note that NP1 = N + 1. The other modifications are self explanatory.

Table 4.1. Modification of Longman's Padé algorithm.

	Original		First Alteration		Second Alteration
	SUBROUTINE PADE(C,D,AL,BE,N)		SUBROUTINE PADE(C,D,AL,BE,N)		SUBROUTINE PADE(C,D,AL,N,NP1)
	DIMENSION C(N),D(N),AL(N,N,N),BE(N,N,N)		DIMENSION C(N),D(N),AL(N,N,N),BE(N,N,N)		DIMENSION C(N),D(N),AL(NP1,N,N)
	D(1) = 1. $D(2) = -C(2)/C(1) $N1 = N-1		D(1) = 1. $D(2) = -C(2)/C(1) $N1 = N-1		D(1) = 1. $D(2) = -C(2)/C(1) $N1 = N-1 $NP3 = N+3 $NP2 = N+2
	DO 5 I = 3,N		DO 5 I = 3,N		DO 5 I = 3,N
	S = C(I)		S = C(I)		S = C(I)
	I1 = I-2		I1 = I-2		I1 = I-2
	DO 6 J = 1,I1		DO 6 J = 1,I1		DO 6 J = 1,I1
6	S = S+C(I-J)*D(J+1)	6	S = S+C(I-J)*D(J+1)	6	S = S+C(I-J)*D(J+1)
5	D(I) = -S	5	D(I) = -S	5	D(I) = -S
	DO 7 J = 1,N				
	DO 7 K = 1,J		DO 7 K = 1,N		DO 7 K = 1,N
			CK = C(K)		CK = C(K)
			DK = D(K)		DK = D(K)
			DO 7 J = K,N		DO 7 J = K,N
	AL(1,J,K) = C(K)		AL(1,J,K) = CK		AL(1,J,K) = CK
7	BE(J,1,K) = D(K)	7	BE(J,1,K) = DK	7	AL(NP1,J,NP1-K) = DK
	DO 10 I = 2,N		DO 10 I = 2,N		DO 10 I = 2,N
	J1 = N+1-I		J1 = N+1-I		J1 = NP1-I
	DO 10 J = 1,J1		DO 10 J = 1,J1		DO 10 J = 1,J1
			BE(J,I,1) = 1.		AL(NP2-I,J,N) = 1.
10	AL(I,J,1) = C(1)	10	AL(I,J,1) = C(1)	10	AL(I,J,1) = C(1)
	DO 11 J = 2,N				
	I1 = N+1-J				
	DO 11 I = 1,I1				
11	BE(I,J,1) = 1.				
	DO 16 I = 2,N1		DO 16 I = 2,N1		DO 16 I = 2,N1
	J1 = N+1-I		J1 = N+1-I		J1 = NP1-I
	DO 16 J = 2,J1		DO 16 J = 2,J1		DO 16 J = 2,J1
					TEMA = AL(I-1,J+1,J+1)/AL(I-1,J,J)
					TEMB = AL(NP3-I,J+1,N-J)/AL(NP3-I,J,NP1-J)
	DO 16 K = 2,J		DO 16 K = 2,J		DO 16 K = 2,J
			BE(J,I,K) = BE(J+1,I-1,K)-BE(J,I-1,K-1)*BE(J+1, I-1,J+1)/BE(J,I-1,J)		AL(NP2-I,J,NP1-K) = AL(NP3-I,J+1,NP1-K)-AL(NP3-I,J,NP2-K)*TEMB
16	AL(I,J,K) = AL(I-1,J+1,K)-AL(I-1,J,K-1)*AL(I-1, J+1,J+1)/AL(I-1,J,J)	16	AL(I,J,K) = AL(I-1,J+1,K)-AL(I-1,J,K-1)*AL(I-1, J+1,J+1)/AL(I-1,J,J)	16	AL(I,J,K) = AL(I-1,J+1,K)-AL(I-1,J,K-1)*TEMA
	DO 17 J = 2,N1				
	I1 = N+1-J				
	DO 17 I = 2,I1				
	DO 17 K = 2,I				
17	BE(I,J,K) = BE(I+1,J-1,K)-BE(I,J-1,K-1)*BE(I+1,				
	RETURN		RETURN		RETURN
	END		END		END

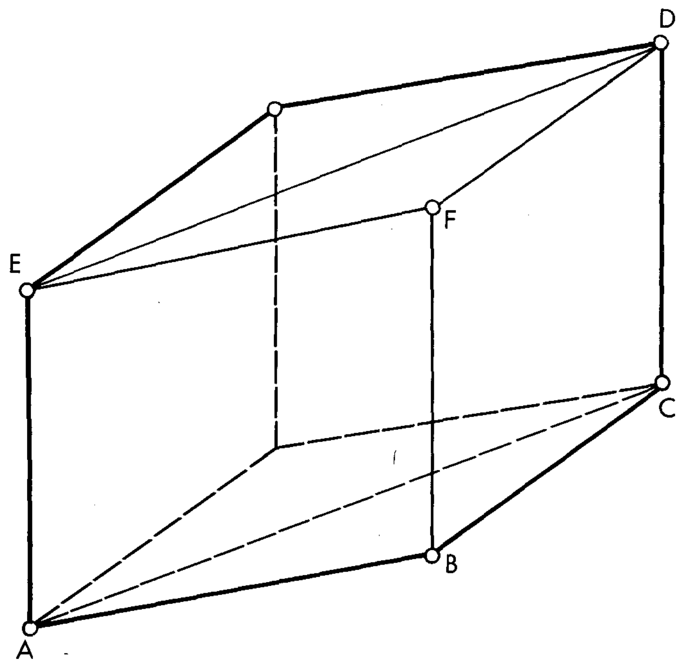


Fig. 4.1. Model of the AL array.

Table 4.2. Modification of Longman's PADVAL algorithm.

Original

```
     SUBROUTINE PADVAL(AL,BE,U,V,W,X,N)
     DIMENSION AL(N,N,N),BE(N,N,N),U(N,N),V(N,N),W(N,N)
     PRINT 16,X
 16  FORMAT (5X,2HX=,E20.8)
     PRINT 15
 15  FORMAT (5X,30H    I    J    W(I,J)    )



     DO 8 I = 1,N
     J1 = N+1-I
     DO 8 J = 1,J1
     U(I,J) = 0.  $  V(I,J) = 0.
     DO 5 K = 1,J
  5  U(I,J) = U(I,J)+AL(I,J,K)*X**(K-1)
     DO 6 K = 1,I
  6  V(I,J) = V(I,J)+BE(I,J,K)*X**(K-1)
     IF(V(I,J))7,9,7
  7  W(I,J) = U(I,J)/V(I,J)
     PRINT 12,I,J,W(I,J)
 12  FORMAT (2I10,E20.8)
     GO TO 8
  9  PRINT 13,I,J
 13  FORMAT (2I10,9X,2H**)
  8  CONTINUE
     RETURN
     END
```

Altered

```
     SUBROUTINE PADVAL(AL,U,V,W,X,XK,N,NP1)
     DIMENSION AL(NP1,N,N),U(N,N),V(N,N),W(N,N),XK(N)
     PRINT 16,X
 16  FORMAT (5X,2HX=,E20.8)
     PRINT 15
 15  FORMAT (5X,30H    I    J    W(I,J)    )
     DO 1 K = 1,N
  1  XK(K) = X**(K-1)
     NP2 = N+2
     DO 8 I = 1,N
     J1 = NP1-I
     DO 8 J = 1,J1
     U(I,J) = 0.  $  V(I,J) = 0.
     DO 5 K = 1,J
  5  U(I,J) = U(I,J)+AL(I,J,K)*XK(K)
     DO 6 K = 1,I
  6  V(I,J) = V(I,J)+AL(NP2-J,I,NP1-K)*XK(K)
     IF(V(I,J))7,9,7
  7  W(I,J) = U(I,J)/V(I,J)
     PRINT 12,I,J,W(I,J)
 12  FORMAT (2I10,E20.8)
     GO TO 8
  9  PRINT 13, I,J
 13  FORMAT (2I10,9X,2H**)
  8  CONTINUE
     RETURN
     END
```

The listing of the original and final versions of PADVAL are displayed in Table 4.2. The changes are straightforward.

Some recent publications concerned with Padé approximants are listed in the references 7, 8, and 9.

References

Chapter 3

1. J.H. Wilkinson and C. Reinsch, Linear Algebra (vol. II of the Handbook for Automatic Computation), Springer-Verlag (New York, 1971).
2. J.M. Boyle, et al., "NATS: A Collaborative Effort to Certify and Disseminate Mathematical Software," to be presented at the 1972 National ACM Conference.
3. "The Certified Eigensystem Package, EISPACK," SIGNUM Newsletter 7, 2 (July 1972), 4-5.

Chapter 4

4. I.M. Longman, "Computation of the Padé Table," Intern. J. Compt. Math. 3, 53 (1971).
5. O. Perron, Die Lehre von dem Kettenbruchen, (Chelsea, 1950), Chap. 10.
6. H.S. Wall, Analytic Theory of Continued Fractions, (Van Nostrand, 1948), p. 388.
7. W.B. Gragg, "The Padé Table and Its Relation to Certain Algorithms of Numerical Analysis," SIAM Review 14, (1972).
8. G.A. Baker, Jr. and J.L. Gammel, The Padé Approximant in Theoretical Physics (Academic Press, 1970).
9. P. Wynn, "The Rational Approximation of Functions which are Formally Defined by a Power Series Expansion," Math. of Computation (1960).

Distribution

LLL Internal Distribution

Roger E. Batzel	
R.P. Abbott	
J.H. Bolstad	
J.L. Brady	
N.W. Davies	
R.P. Dickinson	
S. Fernbach	
F.N. Fritsch	30
C.D. Gardner	
P.H. Gray	
G.L. Hage	
M.A. Harrison	
R.F. Hausman	
N.W. Hetherington	
A.C. Hindmarsh	
B.M. Johnston	
J.I. Karush	
T. Kishi	
V.J. Kransky	
R.M. Lee	
N.K. Madsen	
J.R. Matthews	
S.F. Mendicino	
H.L. Nelson	
R.L. Pexton	10
M.E. Rundquist	2
J.E. Schoonover	
R.E. Shafer	
N.G. Smiriga	
T. Suyehiro	
R.E. von Holdt	10
R.M. Wright	
LBL Library	
TID File	30

External Distribution

R.F. Sincovec
Kansas State University
Manhattan, Kansas

W.R. Cowell
W.J. Cody
Argonne National Laboratory
Argonne, Illinois

J.F. Traub
Carnegie-Mellon University
Pittsburgh, Pennsylvania

L.P. Meissner
University of California
Berkeley, California

B.K. Swartz
Los Alamos Scientific Laboratory
Los Alamos, New Mexico

J.N. Rogers
Sandia Laboratories
Livermore, California

C.W. Gear
University of Illinois
Urbana, Illinois

H.C. Thacher, Jr.
Department of Computer Science
University of Kentucky
Lexington, Kentucky

R.J. Thompson
D.A. Young
Sandia Laboratories
Albuquerque, New Mexico

Technical Information Center,
Oak Ridge, Tennessee 2

HLL:edas