AN ABSTRACT OF THE THESIS OF

KEONG-WAH SUN (Name) for the MASTER OF SCIENCE (Degree)

in ELECTRICAL AND ELECTRONICS ENGINEERING (Major) presented on Sept 25, 1972 (Date)

Title: COMPARISON OF ROUNDING ERRORS, CPU TIME AND MEMORY REQUIREMENTS OF ELEVEN INTEGRATION FORMULAS

Abstract approved: Louis N. Stone

The worst case analysis of rounding error of eleven numerical quadrature formulas on a computer with floating-point arithmetic, base b, and t digit mantissa is calculated. These results are applied to five different integrands using the digital computer CDC 3300 located at Oregon State University. Central processing unit computing time, and memory storage of the computer required by these eleven numerical quadrature formulas are examined also.

Comparison of Rounding Errors, CPU Time and Memory Requirements of Eleven Integration Formulas

by

Keong-Wah Sun

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1973

APPROVED:

Redacted for Privacy

Professor of Electrical and Electronics Engineering

in charge of major

Redacted for Privacy

Head of Department of Electrical and Electronics Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented Sept. 15, 1972

Typed by Clover Redfern for Keong-Wah Sun

ACKNOWLEDGMENTS

I wish to express my appreciation to Dr. Joel Davis for many of his meaningful suggestions and the considerable amount of his time that he spent with me during the preparation of this thesis.

Sincere thanks is also addressed to Professor Louis N. Stone for his continuous guidance towards the accomplishment of this goal.

Finally, I wish to thank the many people who contributed to my advanced study here at Oregon State University. To them, I dedicate whatever is worthwhile in this paper.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

COMPARISON OF ROUNDING ERRORS, CPU TIME AND MEMORY REQUIREMENTS OF ELEVEN INTEGRATION FORMULAS

I. INTRODUCTION

Numerical quadrature, espoused by Newton, Euler, and Gauss, is one of the fields of applied mathematics most markedly affected by the advent of the computer. In the mid-sixties, the introduction of the Romberg and adaptive schemes again led to the systematic generation of quadrature formulas. Throughout the cultivation of quadrature, interest has been concentrated on techniques, convergence, truncation error, stability as well as accuracy.

Today, the usage of quadrature is closely related to digital computers. Unconsciously the knowledge about the effectiveness of integration formulas with respect to digital computers is usually ignored. In applying quadrature formulas, when the usage of an automatic digital computer is essential, and the computation is extensive, the influence of the computer can no longer be ignored. This paper is an effort to bridge the gap between the theoretical formulas of numerical quadrature and the practical world of computing.

The paper is divided into two major parts. The first part is concentrated on developing the theory of the rounding error due to machine representation both binary and decimal numbers, and the worst case analysis of rounding error of eleven numerical integration

formulas by the machine due to repetitive calculation. The result derived is applicable to any size (of bits per word) of digital machine as well as to electronic desk calculators.

The second portion of the paper applies the result of the first part to quadrature of formulae with five different integrands on the digital computer CDC 3300 located at Oregon State University. Time analysis and memory storage of the quadrature required in the computing process are investigated also. Three parameters, rounding error due to computing, central processing unit (CPU) time, and memory storage versus accuracy or total number of accumulated function evaluations (NFUN) are studied also.

II. ROUNDING ERROR[1]

"A computer never makes mistakes" is a familiar slogan for some programmers. But one kind of error that programmers may recognize and always ignore, and which is always caused exclusively by the computer is the rounding error of computing.

2.1. Number[2]

In most of the automatic digital computers, the number representation is classified into two types, one designated as fixed point and the other floating point. A fixed point number consists of two parts, the whole number portion of the number and the fractional part of the number. If only the whole number portion of the number is allowed to be represented while the fractional portion is rounded off and omitted, then the "integer number system" is evolved. In the integer number systems, there is no rounding error involved except overflow.

In the computer, the size of a number is limited by the size of the registers. For example, the decimal number 0.625 can be represented as a binary number without conversion error since it has an

---

[1] Rounding error refers to rounding error contributed by the machine unless specified otherwise. See Wilkinson [7].

[2] Number refers to real number system only.

exact binary equivalent;

$$0.625 = 5/8 = (1/2)^1 + (1/2)^3$$

However, to convert the decimal number 0.626 to its binary equivalent an infinite series is needed;

$$0.626 = (1/2)^1 + (1/2)^3 + (1/2)^{10} + (1/2)^{16} + (1/2)^{17} + (1/2)^{21} + \ldots$$

Assuming that a binary machine has 20 digits available for representing the number, then 0.626 can be read as either

$$0.6259945 \simeq .1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0$$

or

$$0.6260040 \simeq .1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1$$

The difference between the two representations is 2^{-t}, t being the total number of digits and in this case t is 20.

This kind of limitation extends to both fixed-point computation and floating-point computation. We will first discuss briefly the rounding error for fixed-point computation. Then we will introduce the detail analysis of rounding error for floating-point computation.

2.2. Rounding Error in Fixed-Point Computation

The unary operation of the conversion of a real number from base b_1 to another base b_2, such as from decimal to binary,

introduces rounding error as indicated in Section 2. 1. The difference between two adjacent fixed-point numbers in a computer with base b is b^{-t}, while t being the total number of digits of each of the registers in the computer. If the real number is half way between these two adjacent fixed-point numbers, which is the worst case that can ever happen, then the difference between the theoretical representation and the machine representation in base b of the real number in fixed-point mode is $\frac{1}{2} * b^{-t}$.

For addition and subtraction in fixed point, there is no rounding error involved. There is a possibility of overflow.

For multiplication and division, rounding error does play a role. The result for both operation of two t-digit numbers in base b is a number of $2t$ digits in base b. This $2t$ digit resultant is then replaced by a t-digit number in the same base by adding $\frac{1}{2} * b^{-t}$ to the first t digits of the number and truncating the second t digits of the number. The mathematical equations are

$$C_1 = x * y \; ; \tag{2-1}$$

$$C_2 = x / y \; . \tag{2-2}$$

The computational equations are

$$C_1 = x * y + \epsilon \; ; \qquad \text{where } |\epsilon| \leq \frac{1}{2} * b^{-t} \tag{2-3}$$

$$C_2 = x / y + \epsilon \; . \tag{2-4}$$

However, modern machine (computers) usually provide circuitry to handle fractional numbers by "floating-point" arithmetic. For fixed point arithmetic without automatic scaling, there is some gain in speed, but the programmer has to keep track of the scaling. Furthermore, if the programming algorithm is in Fortran, most of the numbers specified will be automatically converted to floating-point before execution of the program. For these reasons, our effort is concentrated on the rounding error of floating-point computations.

2.3. Floating-Point Number

A floating point number X in base b is a collection containing three elements, designated as sign (S), mantissa (M), and exponent (E), representing a real number in the following manner,

$$X = S * M * b^{E} \tag{2-5}$$

where S is the sign of either $+1$ or -1, M is the mantissa, which is a t-digit number such that

$$1/b \leq M < 1,$$

and E is the exponent which is an integer whose magnitude is bounded by a constant dependent on the computer. For example,

Real number	Floating point number In base 10	In base 2
4	$(+1)*(.400)*10^{1}$	$(+1)*(.100000000)*2^{3}$
400	$(+1)*(.400)*10^{3}$	$(+1)*(.110010000)*2^{9}$
-0.625	$(-1)*(.625)*10^{0}$	$(-1)*(.101000000)*2^{0}$

The number 0, is usually represented by zero mantissa and the smallest possible exponent.

In Section 2.1, we mentioned that the limitation of the size of the registers extended to the floating point representation of the real number also. The limitation occurs in the mantissa which allows only a finite number of digits (t) to be represented. It is because of this that most numbers cannot be represented exactly in floating-point.

2.4. Absolute Error

The difference between any two adjacent floating point numbers, used to represent a non-zero real number, is defined to be the gap size. For example, let $X = m*b^{E}$ represent a positive real number with $1/b \leq m < 1$. There exists a M belonging to the interval $[1/b, 1]$ such that $M \leq m \leq M+b^{-t}$. Therefore X can be represented by either $M*b^{E}$ or $(M+b^{-t})*b^{E}$, which ever is closest to X. The difference between these two floating-point numbers or the gap size is then $b^{-t}*b^{E}$. If the real number is half way between these two adjacent floating point numbers, which is the worst case that can

happen, then the difference between the real number and the machine floating point representation is $\frac{1}{2} * b^{-t} * b^{E}$. It is therefore concluded that the absolute error is

$$\begin{aligned} |\text{Error}| &= |\text{Real number} - \text{Floating point representation}| \\ &\leq \frac{1}{2} * b^{-t} * b^{E} \end{aligned} \tag{2-6}$$

2.5. Relative Error

Relative error is defined as the ratio between absolute error and the exact answer. Since the maximum of the absolute value of the rounding error for a floating point representation of a real number is $\frac{1}{2} * b^{-t} * b^{E}$, if the exact representation of the number is $M * b^{E}$, then

$$\begin{aligned} |\text{Max relative error}| &\leq \frac{|\text{Max absolute error}|}{|\text{Exact answer}|} \\ &= \frac{\frac{1}{2} * b^{-t} * b^{E}}{M * b^{E}} \end{aligned} \tag{2-7}$$

which can be simplified to

$$|\text{Relative error}| \leq \frac{\frac{1}{2} * b^{-t}}{M}$$

Since the mantissa of the floating-point number is bounded by

$$1/b \leq M < 1$$

if we assume the worst case when M is $1/b$, then

$$|\text{Relative error}| \leq \frac{\frac{1}{2} * b^{-t}}{1/b}$$

$$= \frac{1}{2} * b^{1-t} \qquad (2\text{-}8)$$

2.6. Accuracy in Conversion of Number From One Base to Another With Carry Over Rounding Error

Now we will take a look at analyzing the number of digits of accuracy we are able to achieve in conversion, such as in the process of converting from base b_2 to base b_1 with the number in b_2 having a rounding error. For example, on the CDC 3300, if we are converting the binary number within the computer to decimal output, let the relative error for the binary number within the computer and the relative error for the decimal output be

$$\frac{\frac{1}{2} * b_1^{-t_1}}{M_1} \quad \text{and} \quad \frac{\frac{1}{2} * b_2^{-t_2}}{M_2}$$

respectively.

Since we are converting the binary number into decimal number, the relative error of the binary number is carried over also. In order to find the effect of this rounding error in the decimal number we form the relationship of

$$\frac{\frac{1}{2} * b_1^{-t_1}}{M_1} \leq \frac{\frac{1}{2} * b_2^{-t_2}}{M_2} \qquad (2\text{-}9)$$

To express t_2 in terms of b_1, b_2, t_1, M_1, and M_2, we rearrange Equation (2-6) and taking the logarithm of the equation. Since in our example b_2 is ten, b_1 is 2, t_1 is 36 digits, and $\log_{10} 10$ is 1, then

$$t_2 = t_1 \log b_1 + \log(M_1/M_2) . \qquad (2\text{-}10)$$

If the mantissa M_1 satisfies

$$\frac{1}{2} \leq M_1 < 1,$$

while M_2 for the decimal machine is

$$.1 \leq M_2 < 1,$$

then for the best case

$$t_2 \leq (36 \log 2 + \log(1/.1)) \quad \text{digits}$$
$$= 11.836 \quad \text{digits} \qquad (2\text{-}11)$$

or in the worst case

$$t_2 \geq (36 \log 2 + \log(.5/1)) \quad \text{digits}$$
$$= 10.535 \quad \text{digits} \qquad (2\text{-}12)$$

Therefore it can be concluded that with a relative rounding error on the 36^{th} bit of the mantissa in a binary floating point number, to be converted to a decimal floating point representation, only ten or eleven

digits of accuracy can be expected.

2.7. Rounding Error in Floating Point Computation

In an ideal computer, with finite word length (number of digits), the machine first performs the operation (+, -, *, /), then it finds the nearest floating point representation of the resultant. The relative rounding error for this ideal machine calculation is bounded by $\frac{1}{2} * b^{1-t}$. The floating point operation for such a machine is

$$Fl(x_1 \circ x_2) = (x_1 \circ x_2)(1+\epsilon) \tag{2-13}$$

where

$$|\epsilon| \leq \frac{1}{2} * b^{1-t}$$

For a nearly optimal computer, the relative rounding error for floating point calculation is bounded by $C_0 * b^{1-t}$, where C_0, a constant imbedded within the computer, is close to 1 and depends on the machine.

In the case of CDC 3300, the computer performs like a nearly ideal machine with b equals to 2, t being 36, and C_0 normally equals to 1 with some exceptional cases in addition and subtraction. For example, a subtraction of two numbers in 4-digit floating-point binary arithmetic in CDC 3300 such as

$$2^5 * (.1000) - 2^4 * (.1111)$$

The subtraction takes place in the form of

$$\begin{array}{r} 2^5*(.1000) \\ \underline{-2^5*(.0111)} \\ 2^5*(.0001) \end{array}$$

The exact sum of this arithmetic is $2^5*(.00001)$. A 50 percent of error is introduced in the computation and $C_0 \gg 1$.

However, a general rule which would also include the exceptional cases of subtraction and addition for floating-point arithmetic can be stated as following,

$$Fl(x_1 \pm x_2) = x_1(1+\epsilon_1) \pm x_2(1+\epsilon_2) \qquad (2-14)$$

where

$$|\epsilon_i| \leq C_0*b^{1-t} \quad i = 1, 2$$

2.8. Repetitive Operation

We now consider the case of accumulation of floating point operations. Considering the mathematical equation:

$$Y = ((X_1+X_2)+X_3)*X_4)$$

For a floating-point computation, the following analysis is executed.

$$
\begin{aligned}
Y_1 &= X_1 \\
Y_2 &= (Y_1+X_2)(1+\epsilon_1) \qquad \text{where} \quad |\epsilon_i| \leq \tfrac{1}{2} * b^{1-t} \\
&= (X_1+X_2)(1+\epsilon_1) \qquad\qquad i = 1, 2, 3 \\
&= X_1(1+\epsilon_1) + X_2(1+\epsilon_1) \\
Y_3 &= (Y_2+X_3)(1+\epsilon_2) \\
&= (X_1(1+\epsilon_1)+X_2(1+\epsilon_1)+X_3)(1+\epsilon_2) \\
&= X_1(1+\epsilon_1)(1+\epsilon_2) + X_2(1+\epsilon_1)(1+\epsilon_2) + X_3(1+\epsilon_2) \\
Y_4 &= (Y_3*X_4)(1+\epsilon_3) \\
&= (X_1(1+\epsilon_1)(1+\epsilon_2)+X_2(1+\epsilon_1)(1+\epsilon_2)+X_3(1+\epsilon_2))*X_4(1+\epsilon_3) \\
&= X_1X_4(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) + X_2X_4(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) \\
&\quad + X_3X_4(1+\epsilon_2)(1+\epsilon_3) \qquad\qquad (2\text{-}15)
\end{aligned}
$$

A. Accumulation of Rounding Error

From Equation (2-15) we see that the rounding errors do accumulate. If we can assume that any power of ϵ smaller than the first power is much smaller than ϵ, which in turn is bounded by $\frac{1}{2} * b^{1-t}$, then the series

$$(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)\ldots(1+\epsilon_N)$$

can be truncated and is approximately equal to

$$1 + \epsilon_1 + \epsilon_2 + \epsilon_3 + \ldots + \epsilon_N.$$

This assumption if valid is the result of the next theorem.

Before we proceed to state the theorem, let us consider the graph of e^x, $(1+x)$, and $(1+1.1x)$ as in Figure 2-1. The following inequalities can be deduced,

$$1+x \leq e^x \leq 1+1.1x \tag{2-16}$$

for $0 \leq x \leq 0.1$.

Theorem (Wilkinson [7]). Let ϵ_i be real for each i and $\sum_{i=1}^{N} |\epsilon_i| \leq .1$, then

$$\prod_{i=1}^{N} (1+\epsilon_i) = 1 + r \tag{2-17}$$

where

$$|r| \leq 1.1 \sum_{i=1}^{N} |\epsilon_i|$$

The proof is as following.

Let

$$\prod_{i=1}^{N} (1+\epsilon_i) = 1 + r$$

therefore r is $\epsilon_1 + \epsilon_2 + \ldots + \epsilon_N + \epsilon_1\epsilon_2 + \epsilon_1\epsilon_3 + \ldots + \epsilon_1\epsilon_2\epsilon_3 \cdots \epsilon_N$

then

$$|r| \leq \prod_{i=1}^{N} (1+|\epsilon_i|) - 1 \tag{2-18}$$

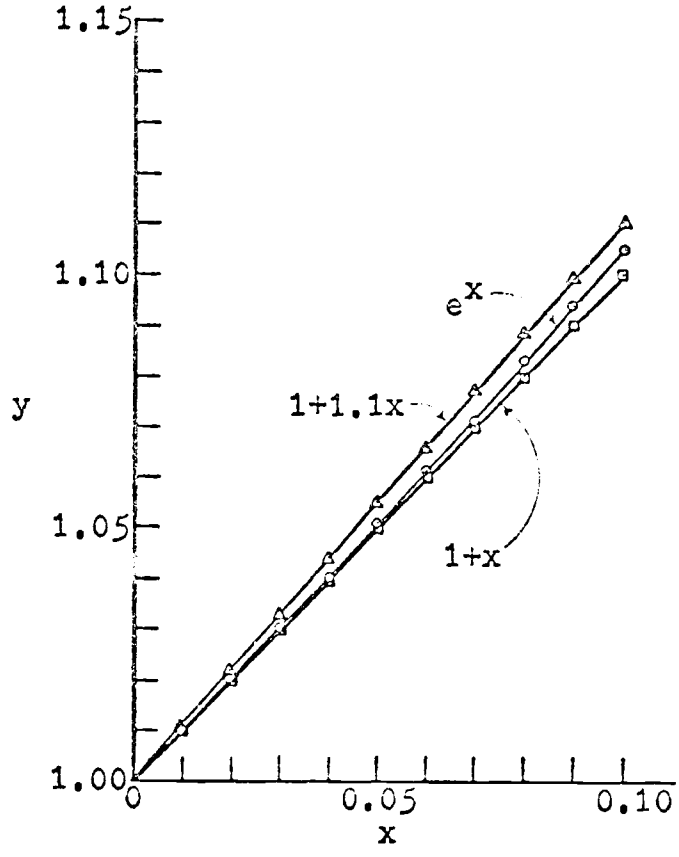


Figure 2-1. The inequalities of $1+x \leq e^x \leq 1+1.1x$ for $0 \leq x \leq 0.1$.

By applying the relationship of the inequalities of Equation (2-16)

and $\sum_{i=1}^{N} |\epsilon_i| \leq .1$

$$|r| \leq \prod_{i=1}^{N} (e^{|\epsilon_i|}) - 1$$

$$= e^{\sum_{i=1}^{N} |\epsilon_i|} - 1$$

$$\leq 1 + 1.1 \sum_{i=1}^{N} \epsilon_i - 1$$

$$\leq 1.1 \sum_{i=1}^{N} |\epsilon_i| \qquad (2\text{-}19)$$

Therefore it can be concluded

$$\prod_{i=1}^{N} (1+\epsilon_i) = 1 + r \leq 1 + |r| \leq 1 + 1.1 \sum_{i=1}^{N} |\epsilon_i| \qquad (2\text{-}20)$$

Since $|\epsilon_i| \leq \frac{1}{2} * b^{1-t}$ for $i+1, 2, 3, \ldots, N$. Then Equation (2-18) is simplified to give

$$|r| \leq 1.1 * N * \frac{1}{2} * b^{1-t} \qquad (2\text{-}21)$$

B. Relative Rounding Error of Repetitive Operation

Again let us rewrite Equation (2-15)

$$Y_4 = X_1X_4(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) + X_2X_4(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)$$
$$+ X_3X_4(1+\epsilon_2)(1+\epsilon_3)$$

with

$$|\epsilon_i| \leq \frac{1}{2} * b^{1-t} \qquad i = 1, 2, 3$$

If Y_4' is the exact value of $((X_1+X_2)+X_3)*X_4$ then the relative rounding error of this repetitive operation is

$$|\text{Error}| = |Y_4 - Y_4'|$$
$$\leq X_1X_4 * 1.1(|\epsilon|+|\epsilon|+|\epsilon|) + X_2X_4 * 1.1(|\epsilon|+|\epsilon|+|\epsilon|)$$
$$+ X_3X_4 * 1.1(|\epsilon|+|\epsilon|) \qquad (2\text{-}22)$$

where, for a nearly optimal computer,

$$|\epsilon| \leq C_0 * b^{1-t}.$$

If we can find a value of M such that

$$|X_i| \leq M \qquad i = 1, 2, 3, 4$$

and let $C_0 = 1$ then

$$|\text{Error}| \leq 1.1 * b^{1-t} * M^2 * (3+3+2)$$
$$\leq 1.1 * b^{1-t} * M^2 * (8) \qquad (2\text{-}23)$$

III. QUADRATURE ROUNDING ERRORS

3.1. Definite Integrals

Let $f(x)$ be a function defined between A and B and reasonably well behaved, i.e., higher order of derivatives exist. If the interval $A \leq X \leq B$ is divided into n subintervals by the points

$$A < X_1 < X_2 \ldots < X_{i-1} < X_i < \ldots < X_{n-1} < B \qquad (3\text{-}1)$$

then let

$$\Delta X_i = X_i - X_{i-1} \qquad (3\text{-}1a)$$

and ξ_i be any point between X_{i-1} and X_i so as to form the sum

$$\sum_{i=1}^{n} f(\xi_i) * \Delta X_i \qquad (3\text{-}1b)$$

Now let the number of intervals n approach infinity in such a manner that all the lengths of the intervals ΔX_i approach zero. Then if the quantity given by expression (3-1b) approaches a limit, this limit is called the definite integrals of $f(X)$ from A to B and is usually denoted by the symbol

$$\int_A^B f(X)\,dx$$

If $f(X)$ has an antiderivative on the closed interval (A, B) then

$$\int_A^B f(X)dX = F(B) - F(A) \qquad (3-2)$$

and the problem is relatively simple to solve. But quite often, the antiderivative of $f(X)$ is not easily available, and many frustrating hours can be spend searching for one. When the antiderivative cannot be found, numerical techniques similar to the definition must be applied in order to find the integral.

Throughout the content of this paper we are going to use approximation in the following form

$$\int_A^B f(X)dX \simeq \sum w_i * f(X_i) \qquad (3-3)$$

where w_i is the coefficients and the abscissas X_i belong to $[A, B]$.

3.2. Rounding Error of Computation in Eleven Quadrature Schemes

In applying definite integrals there are many specific methods. Nevertheless, we shall concentrate our effort on the methods which are widely accepted such as the Newton-Cotes formulae, Romberg's, Simpson's adaptive, and Gaussian methods. Of these, the Newton-Cotes and the Romberg's are methods in which the intervals are chosen in advance. Simpson's adaptive and the Gaussian methods let

the intervals and their location vary according to the analysis. The inherent truncation errors are not discussed here, they are listed with the equations. For detailed discussion see references.

A. Newton-Cotes Formulas (Hildebrand [3])

Of the Newton-Cotes formulas the most widely used are the trapezoidal rule and the celebrated Simpson's rule. The physical interpretation of the trapezoidal rule is that we approximate the graph of the given function by n straight line segments. For Simpson's rule, the graph of the given function is approximated by $n/2$ arcs of parabolas. Simpson's rule is popular because

1. it is exact for polynomials of degree three or less,
2. its coefficients are rational and small,

and 3. it is a rather accurate method for the effort required.

Trapesoidal rule:

$$S = \frac{h}{2}(f(x_0)+f(x_1)) - \frac{h^3}{12} f''(\xi) \qquad (3\text{-}4)$$

in which $A < \xi < B$ and $h = B - A$.

Extended trapezoidal rule (TRAP) (see Figure 3-1):

$$S = h[\frac{1}{2}f(x_0)+f(x_1)+f(x_2)+\ldots+f(x_{N-1})+\frac{1}{2}f(x_N)] + \frac{Nh^3}{12} f''(\xi) \qquad (3\text{-}5)$$

for some ξ in (A, B) and $h = \frac{B-A}{N}$.

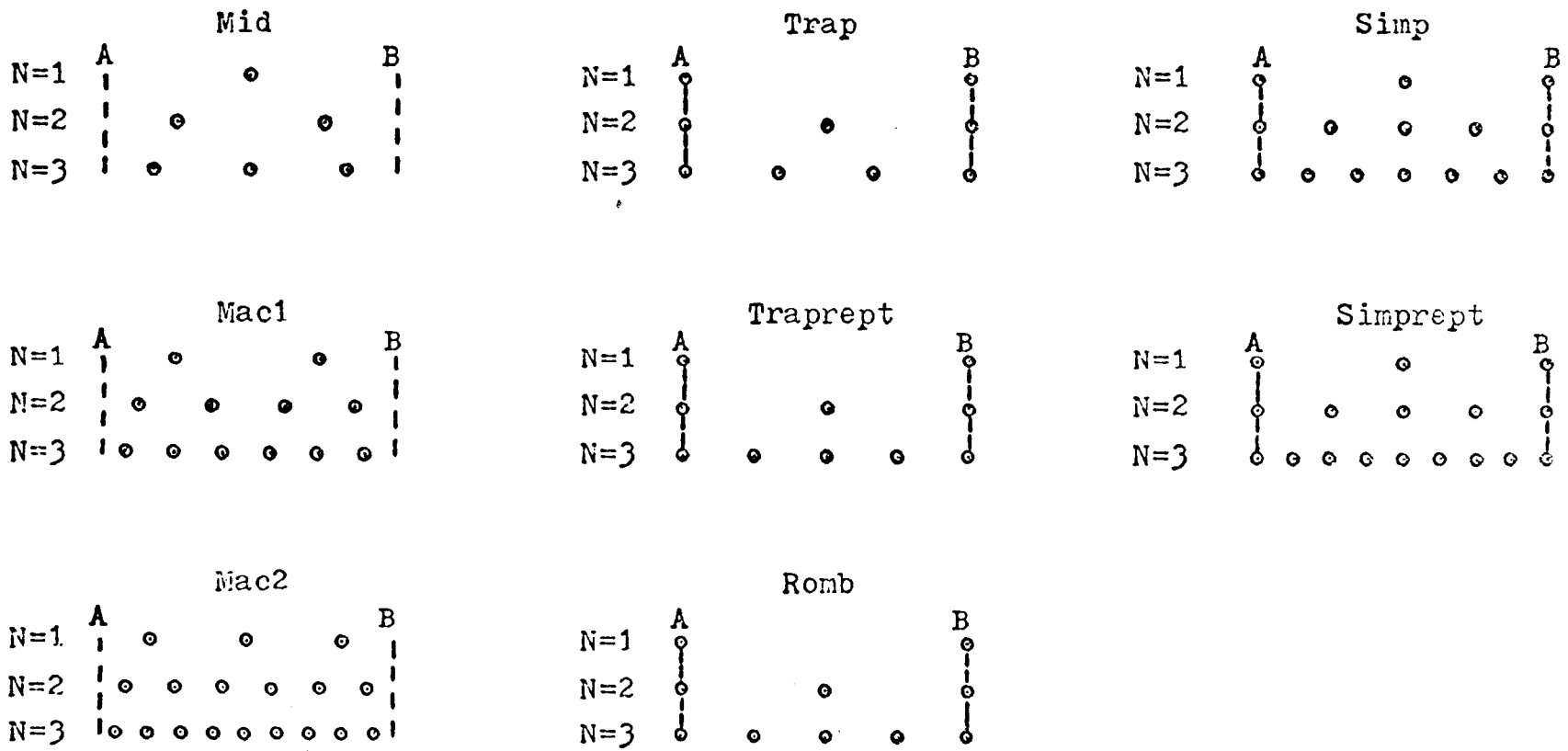


Figure 3-1. Sample of number of function evaluations (NFUN) for Mid, Mac1, Mac2, Trap, Traprept, Romb, Simp and Simprept. (Each o represents a function evaluation within the limit (A, B).)

Simpson's rule:

$$S = \frac{h}{3}(f(x_0)+4f(x_1)+f(x_2)) - \frac{h^5}{90} f^{iv}(\xi) \qquad (3\text{-}6)$$

where

$$A < \xi < B$$

$$h = \frac{B-A}{N}$$

Extended Simpson's rule (SIMP) (see Figure 3-1):

$$S = \frac{h}{3}[f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\ldots+4f(x_{N-3}) + 2f(x_{N-2})+4f(x_{N-1})+f(x_N)] - \frac{Nh^5}{180} f^{iv}(\xi) \qquad (3\text{-}7)$$

for some ξ in (A, B) and $h = \frac{B-A}{N}$

Let us look at the computation rounding error of the trapezoidal rule.

The steps in the program of Appendix I-D for the trapezoidal rule that would contribute to the rounding error in the process of computation are steps 9, 13, 16 and 17, namely,

```
STEP  9  DELTA = (B-A)/N,
STEP 13  SUM = (Y(1)+Y(NN))/2,
STEP 16  SUM = SUM + Y(J),
STEP 17  SUM = SUM * DELTA .
```
(3-8)

(Step 16 is inside of a DO loop.)

There are only two rounding errors for step 9, i.e.,

$$\text{DELTA} = \frac{(B-A)(1+\epsilon_I)}{N}(1+\epsilon_{II}) \tag{3-9}$$

For step 13 the rounding error is

$$\text{SUM} = \frac{(Y(1)+Y(NN))}{2}(1+\epsilon_{III})(1+\epsilon_1) \tag{3-10}$$

NN is N+1, the last point on the abscissas.

Step 16 is in the DO loop. An array of rounding error is compounded. The rounding error accumulates as follows:

$$\text{SUM} = (\text{SUM}+Y(J))(1+\epsilon_J) \qquad J = 2, 3, \ldots, N \tag{3-11}$$

If we let

$$1 + \eta_1 = (1+\epsilon_{III})(1=\epsilon_1) \ldots (1+\epsilon_N)$$

$$1 + \eta_r = (1+\epsilon_r)(1+\epsilon_{r+1}) \ldots (1+\epsilon_N) \qquad r = 2, 3, \ldots, N$$

Then the computational SUM with the rounding error is

$$\text{SUM} = \frac{Y(1)+Y(NN)}{2}(1+\eta_1) + \sum_{J=2}^{N} Y(J)(1+\eta_J) \tag{3-12}$$

In performing step 17, SUM becomes

$$SUM = \left(\frac{Y(1)+Y(NN)}{2}\ (1+\eta_1) + \sum_{J=2}^{N} Y(J)(1+\eta_J)\right)$$

$$*\left(\frac{(B-A)}{N}(1+\epsilon_I)\right)(1+\epsilon_{II})(1+\epsilon_{IV})$$

$$SUM = \frac{Y(1)+Y(NN)}{2}\ (1+\alpha_1) + \sum_{J=2}^{N} Y(J)(1+\alpha_J) \qquad (3\text{-}14)$$

$$1 + \alpha_1 = (1+\eta_1)(1+\epsilon_I)(1+\epsilon_{II})(1+\epsilon_{IV})$$

$$1 + \alpha_J = (1+\eta_J)(1+\epsilon_I)(1+\epsilon_{II})(1+\epsilon_{IV})$$

Equation (3-14) is the computational sum with the rounding error. If we assume an ideal machine exists which would give us an exact sum without any rounding error, and let this sum be designated as SUMT, then the rounding error for the trapezoidal rule is the computational SUM minus SUMT such as:

$$E = SUM - SUMT$$

$$= \left(\frac{Y(1)+Y(NN)}{2}\alpha_1 + \sum_{J=2}^{N} Y_J\alpha_J\right)*DELTA \qquad (3\text{-}15)$$

In taking consideration of only the first order of ϵ, Equation (3-15) can be rewritten as

$$|E| \leq \left(\frac{|Y(1)+Y(NN)|}{2}\ |\alpha_1| + \sum_{J=2}^{N} |Y_J|\,|\alpha_J|\right)*DELTA \qquad (3\text{-}16)$$

where

$$|a_1| \leq (1.1)(N+4)*b^{1-t} \qquad \text{by Theorem 1}$$

$$|a_J| \leq (1.1)(N-J+1+3)*b^{1-t} \qquad \text{by Theorem 1}$$

$$= (1.1)(N-J+4)*b^{1-t}$$

substituting a's into the equation, Equation (3-16) becomes

$$|E| \leq DELTA*(1.1)*b^{1-t}*\left(\frac{|Y(1)+Y(NN)|}{2}(N+4)+\sum_{J=2}^{N}|Y_J|(N-J+4)\right) \tag{3-17}$$

If we assume

$$|Y(J)| \leq M \qquad J = 2, 3, 4, \ldots, N$$

$$|Y(1)| \leq M$$

and

$$|Y(NN)| \leq M$$

then

$$|E| \leq DELTA*(1.1)*b^{1-t}*M*\left((N+4)+\sum_{J=2}^{N}(N-J+4)\right)$$

$$= DELTA*(1.1)*b^{1-t}*M*\left(N+4+\frac{4+N+2}{2}(N-2+1)\right)$$

$$= DELTA*(1.1)*b^{1-t}*M*\left(\frac{2N+8+(N+6)(N-1)}{2}\right)$$

$$= DELTA*(1.1)*b^{1-t}*M*\left(\frac{2N+8+N^2+5N-6}{2}\right)$$

$$= DELTA*N*(1.1)*b^{1-t}*M*\left(\frac{N^2+7N+2}{2N}\right)$$

$$= (B-A)*(1.1)*b^{1-t}*M*\left(\frac{N^2+7N+2}{2N}\right) \tag{3-18}$$

If we designate

$$\delta = (B-A)*(1.1)*b^{1-t}*M \tag{3-19}$$

then the rounding error is

$$|E| \leq \delta * \text{Rounding error coefficient}$$

where the formula for calculating the rounding error coefficient of the trapezoidal rule is calculated as above;

$$\text{Rounding error coefficient} = \frac{N^2+7N+2}{2N} \qquad N = 1, 2, 3, \ldots$$

For Simpson's rule (SIMP) (see Figure 3-1), if $N = 1$

$$S = \frac{h}{3}[f(x_0)+4f(x_1)+f(x_2)] \tag{3-20}$$

the rounding error is

$$|E| \leq (B-A)*(1.1)*b^{1-t}*M*(7) \tag{3-21}$$

if $N = 2$

$$S = \frac{h}{3}[f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+f(x_4)] \tag{3-22}$$

the rounding error is expressed as

$$|E| \leq (B-A)*(1.1)*b^{1-t}*M*(25.5/3) \tag{3-23}$$

for $N \geq 3$, the general expression of rounding error for Simpson's rule is

$$|E| \leq (B-A)*(1.1)*b^{1-t}*M*(\frac{3N^2+94N-16}{12N}) \qquad (3-24)$$

Instead of exploring higher degrees of Newton-Cotes formulas, which have large coefficients and emphasis the rounding errors, we examine composite rules of trapezoidal rule, and Simpson's rule such as in Section B.

B. Repeated Trapezoidal (TRAPREPT) and Repeated Simpson (SIMPREPT) (see Figure 3-1)

Let us reconsider the trapezoidal and Simpson's rule. Instead of dividing the interval by consecutive numbers (N), we choose to bisect the interval and subintervals each time.

The interval (h) for trapezoidal will become 1, $\frac{1}{2}$, $\frac{1}{4}$ of (B-A) and the number of function evaluation increases each time, such as 2 points, 3 points, and 5 points. For the Simpson's method, the intervals will be $\frac{1}{2}$, $\frac{1}{4}$ or $\frac{1}{8}$, of (B-A) and the number of function evaluation points is 3, 5, 9 etc. In essence each of the schemes is applied to the interval, and if it is not adequate, the interval again is divided and the formula is applied twice as often. If the agreement is not good enough, the process is repeated. These schemes are called the repeated trapezoidal and the repeated Simpson's method. Their rounding errors are as follows:

for the repeated trapezoidal

$$|E| \leq (B-A)*(1.1)*b^{1-t}*M*(-\frac{11}{3}(\frac{1}{2})^{N-1}+\frac{1}{12}2^{N}+N+\frac{11}{2}) \quad (3-25)$$

$$N = 1, 2, 3, 4, \ldots$$

for the repeated Simpson

$$|E| \leq (B-A)*(1.1)*b^{1-t}*M*(2^{2N}(\frac{7}{12})+2^{N}(3N+\frac{37}{2})-2N-\frac{79}{12})/(2^{N}*3)$$

$$N= 1, 2, 3, 4, \ldots \quad (3-26)$$

C. Mid-Point Rules (MID)

The mide-point rule (see Davis and Rabinowitz [2]) or the rectangular rule is one of the oldest, simplest, and most direct schemes that has been derived. It is only more complicated than counting squares. However, the virtues have seemed to be long forgotten with the glories of classical numerical analysis. The mid-point rule, like the trapezoidal rule, is exact for linear functions. The equation is

$$S = hf(A+\frac{1}{2}h)+\frac{h^3}{24}f''(\xi) \quad (3-27)$$

$$A < \xi < B$$

$$h = B - A$$

The extended mid-point rule (see Figure 3-1) is

$$S = h \sum_{K=0}^{N-1} f(A+\frac{2K-1}{2} h) + \frac{h^3}{24N^2} f''(\xi) \qquad (3\text{-}28)$$

$$A < \xi < B$$

$$h = \frac{B-A}{N}$$

Again let us take a close look at the effect of rounding errors in the computation of the mid-point rule.

The steps from the Fortran program (Appendix I-A) that are involved are

$$\text{STEP 7} \qquad \text{DELTA} = \frac{B-A}{N};$$

and step 13 is in the DO loop,

$$\text{STEP 13} \qquad \text{SUM} = \text{SUM} + Y(I) \qquad I = 1, 2, \ldots, N$$

$$\text{STEP 15} \qquad \text{SUM} = \text{SUM} * \text{DELTA} \qquad (3\text{-}29)$$

Step 7 has only two rounding errors, one from subtracting A from B and the other from dividing the result by N

$$\text{DELTA} = \frac{(B-A)}{B} (1+\epsilon_I)(1+\epsilon_{II}) \qquad (3\text{-}30)$$

Step 13 is in the DO loop with subscript bound N, there is no

rounding error the very first time when SUM = 0, i.e., SUM = 0 + Y(1). The accumulation of the rounding error is a series as

$$SUM = \sum_{J=1}^{N} Y(J)(1+\eta_J) \tag{3-31}$$

where

$$(1+\eta_1) = \prod_{i=2}^{N} (1+\epsilon_i)$$

$$(1+\eta_2) = \prod_{i=2}^{N} (1+\epsilon_i)$$

$$(1+\eta_3) = \prod_{i=3}^{N} (1+\epsilon_i)$$

$$(1+\eta_r) = \prod_{i=r}^{N} (1+\epsilon_i)$$

For step 15 there is only one rounding error of multiplying SUM by DELTA. Therefore

$$SUM = \sum_{J=1}^{N} Y(J)(1+\eta_J) * \frac{(B-A)}{N} (1+\epsilon_I)(1+\epsilon_{II})(1+\epsilon_{III}) \tag{3-32}$$

Let

$$1 + \alpha_1 = (1+\eta_1)(1+\epsilon_I)(1+\epsilon_{II})(1+\epsilon_{III})$$

$$1 + \alpha_2 = (1+\eta_2)(1+\epsilon_I)(1+\epsilon_{II})(1+\epsilon_{III})$$

$$1 + \alpha_r = (1+\eta_r)(1+\epsilon_I)(1+\epsilon_{II})(1+\epsilon_{III}) \quad r = 3, 4, \ldots, N$$

Then

$$SUM = \sum_{J=1}^{N} Y(J)(1+a_J) * \frac{(B-A)}{N}, \tag{3-33}$$

Again if an exact answer is available as SUMT then the rounding error is

$$|E| = SUM - SUMT$$

$$= \sum_{J=1}^{N} Y(J)a_J \frac{(B-A)}{N} \tag{3-34}$$

as

$$\begin{aligned} |a_1| &\leq 1.1(N-1+3)*b^{1-t} \quad \text{by Theorem 1} \\ |a_2| &\leq 1.1(N-1+3)*b^{1-t} \\ |a_r| &\leq 1.1(N-r+1+3)*b^{1-t} \quad r = 3, 4, 5, \ldots, N \end{aligned} \tag{3-34a}$$

Then assuming $|Y(J)| \leq M$ for $J = 1, 2, 3, \ldots, N$ and substituting a's by Equation (3-34a), Equation (3-34) is re-written as

$$|E| \leq \frac{(B-A)}{N} (1.1)b^{1-t} M((N+2)+\sum_{J=2}^{N} (N-J+4))$$

$$= \frac{(B-A)}{N} (1.1)b^{1-t} M((N+2)+ \frac{N+2+4}{2} (N-2+1))$$

$$= (B-A)\ (1.1)b^{1-t} M(\frac{N^2+7N-2}{2N}) \tag{3-35}$$

$$N = 1, 2, 3, \ldots$$

D. Maclaurin Integration Formulas

In the same family with mid-point rule are the Maclaurin integration formulae (MAC1 and MAC2, see Figure 3-1) (Hildebrand [3]).

$$\text{(MAC1)} \qquad S = h\left(f\left(A+\frac{1}{2}h\right) + f\left(A+\frac{3}{2}h\right)\right) + \frac{h^3}{12}f^{ii}(\xi), \tag{3-36}$$

$$h = \frac{B-A}{2}, \quad A < \xi < B$$

$$\text{(MAC2)} \qquad S = \frac{3h}{8}\left(3f\left(A+\frac{1}{2}h\right)+2f\left(A+\frac{3}{2}h\right)+3f\left(A+\frac{5}{2}h\right)\right)$$

$$+ \frac{21h}{640} f^{iv}(\xi) , \tag{3-37}$$

$$h = \frac{B-A}{3}, \quad A < \xi < B$$

Equation (3-36) is obtained by dividing the interval into two equal parts and applying the mid-point rule to each subinterval. The extended scheme of this rule is to divide the interval by $2*N$, $N = 1, 2, 3, \ldots$. For $N = 1$, two points are evaluated. For $N = 2$, four points are considered, $N = 3$, 6 points are involved (see Figure 3-1). The same repetitive strategy is applied to obtain Equation (3-37) except the interval is divided by $3*N$, $N = 1, 2, 3, \ldots$. If $N = 1$, 3 points are evaluated, for $N = 2$, 6 points are calculated (see Figure 3-1). The rounding error for these schemes are

for Equation (3-36)

$$|E| \leq (B-A)(1.1)b^{1-t}M\left(\frac{4N^2+14N-2}{N}\right) \qquad (3\text{-}38)$$

and for Equation (3-37)

$$|E| \leq (B-A)(1.1)b^{1-t}M\left(\frac{14N^2+112N-10}{16N}\right) \qquad (3\text{-}39)$$

E. Romberg's Method (ROMB)

The Romberg method was described in 1955 [1, 4]. The scheme is based solely on the trapezoidal rule and extrapolation. It converges when approximating the integral of a continuous function in a given interval. It is based upon a sequence of trapezoidal rule estimations for the area, each estimate halving the h of the preceding one (see Figure 3-1). These are labelled in the first column as

$$T_{11}$$

$$T_{12}$$

$$T_{13}$$

$$T_{14}$$

where

$$T_{1,k} = h\left(\frac{1}{2}f(A)+f(A+h)+\ldots+\frac{1}{2}f(B)\right) \qquad (3\text{-}40)$$

and

$$h = \frac{B-A}{2^{k-1}} \quad \text{and} \quad k = 1, 2, 3\ldots$$

Then from this we construct a triangular array of convergents by the rule

$$T_{\ell,k} = \frac{4^{(\ell-1)}T_{\ell-1,k+1} - T_{\ell-1,k}}{4^{(\ell-1)}-1} \tag{3-41}$$

with

$$\ell = 2, 3, 4, \ldots \quad \text{and} \quad k = 1, 2, 3, \ldots$$

A tableau is formed as in Table 3-1.

Table 3-1. Romberg's tableau.

T_{11}	T_{21}	T_{31}	T_{41}
T_{12}	T_{22}	T_{32}	
T_{13}	T_{23}		
T_{14}			

The truncation error of $T_{\ell,k}$ is

$$E = \frac{(B-A)h^{2\ell}2^{\ell^2-\ell}B'_{2\ell}f^{2\ell}(\xi)}{(2\ell)!} \tag{3-42}$$

with h, as the step size in the final, smallest subdivision, that is

$$h = \frac{B-A}{2^{\ell+k-2}}$$

and

$$\ell = 1, 2, 3\ldots; \quad k = 1, 2, 3$$

The quantity $B'_{2\ell}$ is the (2ℓ)th Bernoulli number, i.e., $B'_2 = 1/6$, $B'_4 = -1/30$, and $B'_6 = 1/42$; $f^{2\ell}(\xi)$ is the (2ℓ)th derivative of the function and ξ is some point between A and B.

The truncation error term expressing the power of h increases by two each time we go one column to the right, while h itself halves each row downward.

In applying Romberg's scheme, one has to be careful in examining the steps in between, especially for periodic functions. In comparing the extrapolated steps in between, such as T_{11} and T_{21}, specifying the tolerance may not be sufficient and thus an erroneous exit from the program may occur.

The rounding error coefficients for Romberg's rule is manipulated by inserting the following formulae

```
ET(1,1) = 3
ET(1,k+1) = (-11/3*0.5**K+2.**(K+1)/12.+(K+1)+11./2.)
ET(N,M) = ET(N-1,M+1)*4./3.+ET(N-1,M)/3.+1.          (3-43)
```

after steps 10, 20, and 26 respectively in the Fortran program of Appendix I-F, such as in Appendix I-G. The rounding error of $T_{\ell,k}$ is ET(N,M) multiplied by δ where

$$\delta = (B-A)*(1.1)*b^{1-t}*M$$

Table 3-2 gives the results of the rounding error coefficients for the Romberg method with a dimension of (15, 15). The rounding error of the method is bounded by these coefficients multiplied by δ where $\delta = (B-A)(1.1)b^{1-t}M$

F. Simpson's Adaptive Method (SIMPAD)

Simpson's Adaptive method applies Simpson's three-point and seven-point quadrature repeatedly (see Figure 3-2). The interval is divided into three sections. Each section is evaluated by the seven points quadrature which is a repetition of Simpson's three point rule three times.

The interval (A, B) is considered as the first level. This is divided into three equal sections as $(A, A+\frac{h}{3})$, $(A+\frac{h}{3}, A+\frac{2h}{3})$, and $(A+\frac{2h}{3}, A+h)$. For each of these three sections Simpson's three point rule is applied first.

Simpson's three ponts rule

$$R(A, A+h)f(x) = \frac{h}{6}(f(A)+4f(A+\frac{h}{2})+f(A+h)) \qquad (3-44)$$

Then Simpson's seven points quadrature is applied to the same section;

Simpson's seven points quadrature

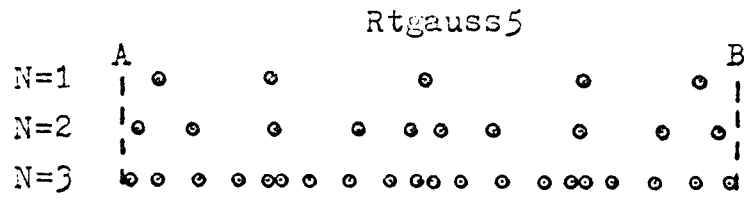
$$R^3(A, A+h)f(x) = R(A, A+\frac{h}{3})f(x)+R(A+\frac{h}{3}, A+\frac{2h}{3})f(x)+R(A+\frac{2h}{3}, A+h)f(x) \qquad (3-45)$$

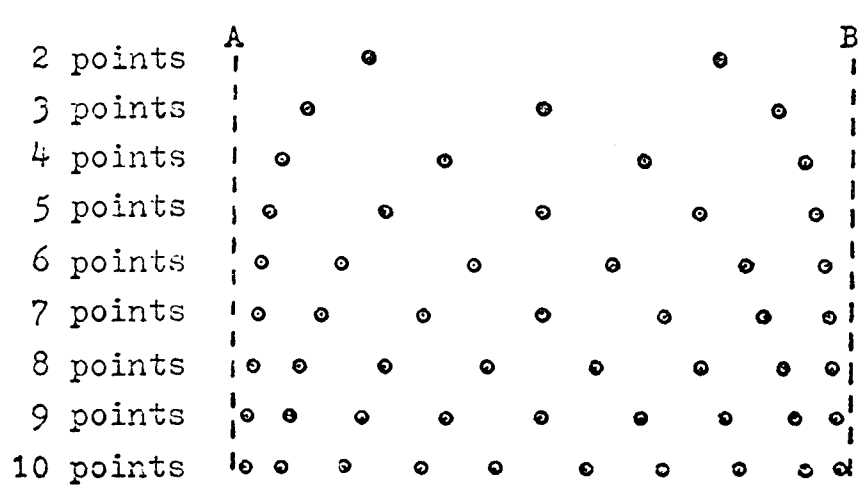
Table 3-2. Rounding error coefficients of Romberg's method.

(1, 1)=3.00E 00	(2, 1)=1.00E 01	(3, 1)=2.30E 01	(4, 1)=4.75E 01	(5, 1)=9.48E 01
(1, 2)=6.00E 00	(2, 2)=1.40E 01	(3, 2)=2.91E 01	(4, 2)=5.84E 01	(5, 2)=1.18E 02
(1, 3)=9.25E 00	(2, 3)=1.76E 01	(3, 3)=3.58E 01	(4, 3)=7.30E 01	(5, 3)=1.52E 02
(1, 4)=1.04E 01	(2, 4)=2.17E 01	(3, 4)=4.50E 01	(4, 4)=9.61E 01	(5, 4)=2.16E 02
(1, 5)=1.29E 01	(2, 5)=2.76E 01	(3, 5)=5.00E 01	(4, 5)=1.37E 02	(5, 5)=3.32E 02
(1, 6)=1.57E 01	(2, 6)=3.74E 01	(3, 6)=8.59E 01	(4, 6)=2.14E 02	(5, 6)=5.55E 02
(1, 7)=2.31E 01	(2, 7)=5.51E 01	(3, 7)=1.38E 02	(4, 7)=3.62E 02	(5, 7)=9.95E 02
(1, 8)=3.48E 01	(2, 8)=8.89E 01	(3, 8)=2.37E 02	(4, 8)=6.55E 02	(5, 8)=1.87E 03
(1, 9)=5.72E 01	(2, 9)=1.54E 02	(3, 9)=4.31E 02	(4, 9)=1.24E 03	(5, 9)=3.62E 03
(1,10)=1.01E 02	(2,10)=2.84E 02	(3,10)=8.18E 02	(4,10)=2.39E 03	(5,10)=7.07E 03
(1,11)=1.87E 02	(2,11)=5.42E 02	(3,11)=1.59E 03	(4,11)=4.70E 03	(5,11)=1.40E 04
(1,12)=3.59E 02	(2,12)=1.06E 03	(3,12)=3.13E 03	(4,12)=9.31E 03	
(1,13)=7.01E 02	(2,13)=2.08E 03	(3,13)=6.20E 03		
(1,14)=1.38E 03	(2,14)=4.13E 03			
(1,15)=2.75E 03				

(6, 1)=1.90E 02	(7, 1)=3.91E 02	(8, 1)=8.45E 02	(9, 1)=1.95E 03	(10, 1)=4.78E 03
(6, 2)=2.45E 02	(7, 2)=5.35E 02	(8, 2)=1.25E 03	(9, 2)=3.10E 03	(10, 2)=8.17E 03
(6, 3)=3.40E 02	(7, 3)=8.01E 02	(8, 3)=2.01E 03	(9, 3)=5.35E 03	(10, 3)=1.49E 04
(6, 4)=5.15E 02	(7, 4)=1.31E 03	(8, 4)=3.51E 03	(9, 4)=9.78E 03	(10, 4)=2.80E 04
(6, 5)=9.52E 02	(7, 5)=2.30E 03	(8, 5)=6.46E 03	(9, 5)=1.86E 04	(10, 5)=5.44E 04
(6, 6)=1.51E 03	(7, 6)=4.27E 03	(8, 6)=1.23E 04	(9, 6)=3.61E 04	(10, 6)=1.07E 05
(6, 7)=2.82E 03	(7, 7)=8.18E 03	(8, 7)=2.40E 04	(9, 7)=7.12E 04	
(6, 8)=5.43E 03	(7, 8)=1.60E 04	(8, 8)=4.74E 04		
(6, 9)=1.06E 04	(7, 9)=3.15E 04			
(6,10)=2.10E 04				

(11, 1)=1.25E 04	(12, 1)=3.42E 04	(13, 1)=9.67E 04	(14, 1)=2.80E 05	(15, 1)=8.22E 05
(11, 2)=2.25E 04	(12, 2)=6.40E 04	(13, 2)=1.86E 05	(14, 2)=5.46E 05	
(11, 3)=4.23E 04	(12, 3)=1.23E 05	(13, 3)=3.63E 05		
(11, 4)=8.19E 04	(12, 4)=2.42E 05			
(11, 5)=1.61E 05				





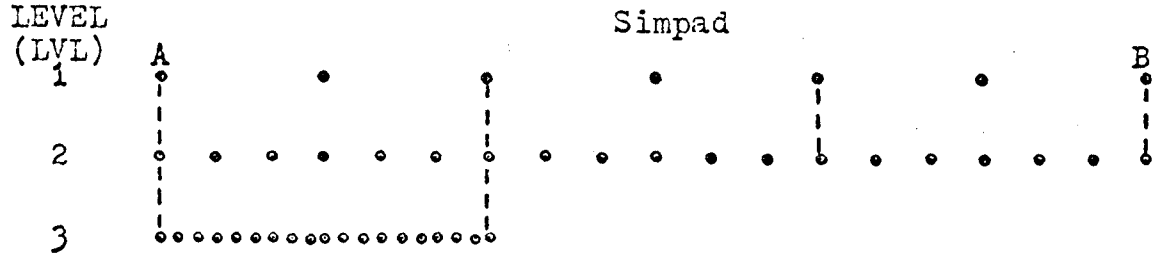


Figure 3-2. Sample of number of function evaluations (NFUN) for Rtgauss5, Gauss, and Simpad. (Each ⊙ represents a function evaluation within the limit (A, B).)

If the Simpson's seven point quadrature does not agree adequately with the three point rule, then the region is divided into three subintervals again and the seven points quadrature is repeated to each of the subintervals. This part of the method is recursive.

In so doing, computational labor is saved; for no further work is expended on the parts where the area is already adequately estimated.

The accuracy tolerance between stages (or prescribed accuracy) is

$$\gamma(r) = \frac{\gamma}{(\sqrt{3})^r} \tag{3-46}$$

where γ is the expected final tolerance and r is level (Lyness [5], McKeeman [6]).

The rounding error coefficients of a certain interval (I) at a certain level (LVL), is TESUM, where

$$\text{TESUM} = \text{ESUM(LVL, 1)} + \text{ESUM(LVL, 2)} + \text{ESUM(LVL, 3)} \tag{3-47}$$

ESUM(LVL, L), L = 1, 2, and 3, is the rounding error coefficients in calculating the integral of f(x) over the (L)th subinterval of (I). For each L, there are two cases. If further subdivision of the (L)th subinterval is not necessary, then

$$\text{ESUM(LVL, L)} = (18.*\text{CLVL}+104)/(6*3*\text{CLVL}) \tag{3-48}$$

where

$$CLVL = LVL$$

Otherwise, ESUM(LVL, L) is the TESUM for the (L)th subinterval. The rounding error is the last calculated TESUM multiplied by δ where

$$\delta = (B-A)*(1.1)*b^{1-t}*M$$

An example of the rounding error coefficients of Simpson's adaptive method for $\int_0^1 \frac{1}{\sqrt{x}}\,dx$ is shown in Table 3-3.

G. Gaussian Method (GAUSS) [2, 3]

The Gaussian quadrature gives up equal spacings of the ordinates and permits their location to be chosen so as to optimize the estimation of the area. The ordinates actually turn out to be the roots of the Legendre polynomial of appropriate degree. The approximation can be expressed as follows:

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} w_i f(x_i) + R_n \qquad (3\text{-}49)$$

in which x_i is the ith zero of $P_n(x)$, the nth Legendre polynomial and value of w_i as given by

$$w_i = \frac{2}{nP_{n-1}(x_i)P'_n(x_i)} \qquad (3\text{-}50)$$

Table 3-3. Rounding error coefficients of Simpson's adaptive method for $\int_0^1 \frac{1}{\sqrt{x}}\,dx$.

ESUM =	5.213E-13	LVL =	29	L = 1
ESUM =	5.213E-13	LVL =	29	L = 2
ESUM =	5.213E-13	LVL =	29	L = 3
TESUM=	5.00E 00			
ESUM =	5.000E 00	LVL =	28	L = 1
ESUM =	1.520E-12	LVL =	28	L = 2
ESUM =	1.520E-12	LVL =	28	L = 3
TESUM=	1.00E 01			
ESUM =	1.000E 01	LVL =	27	L = 1
ESUM =	4.430E-12	LVL =	27	L = 2
ESUM =	4.430E-12	LVL =	27	L = 3
TESUM=	1.50E 01			
ESUM =	1.500E 01	LVL =	26	L = 1
ESUM =	1.290E-11	LVL =	26	L = 2
ESUM =	1.290E-11	LVL =	26	L = 3
TESUM=	2.00E 01			
ESUM =	2.000E 01	LVL =	25	L = 1
ESUM =	3.751E-11	LVL =	25	L = 2
ESUM =	3.751E-11	LVL =	25	L = 3
TESUM=	2.50E 01			
ESUM =	2.500E 01	LVL =	24	L = 1
ESUM =	1.090E-10	LVL =	24	L = 2
ESUM =	1.090E-10	LVL =	24	L = 3
TESUM=	3.00E 01			
ESUM =	3.000E 01	LVL =	23	L = 1
ESUM =	3.163E-10	LVL =	23	L = 2
ESUM =	3.163E-10	LVL =	23	L = 3
TESUM=	3.50E 01			
ESUM =	3.500E 01	LVL =	22	L = 1
ESUM =	9.177E-10	LVL =	22	L = 2
ESUM =	9.170E-10	LVL =	22	L = 3
TESUM=	4.00E 01			
ESUM =	4.000E 01	LVL =	21	L = 1
ESUM =	2.656E-09	LVL =	21	L = 2
ESUM =	2.656E-09	LVL =	21	L = 3
TESUM=	4.50E 01			
ESUM =	4.500E 01	LVL =	20	L = 1
ESUM =	7.630E-09	LVL =	20	L = 2
ESUM =	7.630E-09	LVL =	20	L = 3
TESUM=	5.00E 01			
ESUM =	5.000E 01	LVL =	19	L = 1
ESUM =	2.218E-08	LVL =	19	L = 2
ESUM =	2.218E-08	LVL =	19	L = 3
TESUM=	5.50E 01			
ESUM =	5.500E 01	LVL =	18	L = 1
ESUM =	6.396E-08	LVL =	18	L = 2
ESUM =	6.396E-08	LVL =	18	L = 3
TESUM=	6.00E 01			
ESUM =	6.000E 01	LVL =	17	L = 1
ESUM =	1.841E-07	LVL =	17	L = 2
ESUM =	1.841E-07	LVL =	17	L = 3
TESUM=	6.50E 01			
ESUM =	6.500E 01	LVL =	16	L = 1
ESUM =	5.291E-07	LVL =	16	L = 2
ESUM =	5.291E-07	LVL =	16	L = 3
TESUM=	7.00E 01			
ESUM =	7.000E 01	LVL =	15	L = 1
ESUM =	1.518E-06	LVL =	15	L = 2
ESUM =	1.518E-06	LVL =	15	L = 3
TESUM=	7.50E 01			
ESUM =	7.500E 01	LVL =	14	L = 1
ESUM =	4.344E-06	LVL =	14	L = 2
ESUM =	4.344E-06	LVL =	14	L = 3
TESUM=	8.00E 01			
ESUM =	8.000E 01	LVL =	13	L = 1
ESUM =	1.241E-05	LVL =	13	L = 2
ESUM =	1.241E-05	LVL =	13	L = 3
TESUM=	8.50E 01			
ESUM =	8.500E 01	LVL =	12	L = 1
ESUM =	3.533E-05	LVL =	12	L = 2
ESUM =	3.533E-05	LVL =	12	L = 3
TESUM=	9.00E 01			
ESUM =	9.000E 01	LVL =	11	L = 1
ESUM =	1.004E-04	LVL =	11	L = 2
ESUM =	1.004E-04	LVL =	11	L = 3
TESUM=	9.50E 01			
ESUM =	9.500E 01	LVL =	10	L = 1
ESUM =	2.841E-04	LVL =	10	L = 2
ESUM =	2.841E-04	LVL =	10	L = 3
TESUM=	1.00E 02			
ESUM =	1.000E 02	LVL =	9	L = 1
ESUM =	8.016E-04	LVL =	9	L = 2
ESUM =	8.016E-04	LVL =	9	L = 3
TESUM=	1.05E 02			
ESUM =	1.050E 02	LVL =	8	L = 1
ESUM =	2.252E-03	LVL =	8	L = 2
ESUM =	2.252E-03	LVL =	8	L = 3
TESUM=	1.10E 02			
ESUM =	1.100E 02	LVL =	7	L = 1
ESUM =	6.300E-03	LVL =	7	L = 2
ESUM =	6.300E-03	LVL =	7	L = 3
TESUM=	1.15E 02			
ESUM =	1.150E 02	LVL =	6	L = 1
ESUM =	1.753E-02	LVL =	6	L = 2
ESUM =	1.753E-02	LVL =	6	L = 3
TESUM=	1.20E 02			
ESUM =	1.201E 02	LVL =	5	L = 1
ESUM =	4.847E-02	LVL =	5	L = 2
ESUM =	4.847E-02	LVL =	5	L = 3
TESUM=	1.25E 02			
ESUM =	1.252E 02	LVL =	4	L = 1
ESUM =	1.331E-01	LVL =	4	L = 2
ESUM =	1.331E-01	LVL =	4	L = 3
TESUM=	1.30E 02			
ESUM =	1.304E 02	LVL =	3	L = 1
ESUM =	3.621E-01	LVL =	3	L = 2
ESUM =	3.621E-01	LVL =	3	L = 3
TESUM=	1.36E 02			
ESUM =	1.361E 02	LVL =	2	L = 1
ESUM =	9.753E-01	LVL =	2	L = 2
ESUM =	9.753E-01	LVL =	2	L = 3
TESUM=	1.43E 02			
ESUM =	1.431E 02	LVL =	1	L = 1
ESUM =	2.593E 00	LVL =	1	L = 2
ESUM =	2.593E 00	LVL =	1	L = 3
TESUM=	1.53E 02			

The Legendre polynomials are calculated by the recursion relations such as

$$P_0(x) = 1$$

$$P_1(x) = 1$$

$$P_2(x) = \frac{1}{2}(3x^2 - 1)$$

$$\vdots$$

$$P_{n+1}(x) = \frac{2n+1}{n+1} xP_n(x) - \frac{n}{n+1} P_{n-1}(x) \qquad (3\text{-}51)$$

where

$$\int_{-1}^{1} P_m(x)P_n(x)dx = \begin{cases} 0 & m \neq n \\ 2/2n+1 & m = n \end{cases}$$

and the error term R_n is given by

$$R_n = \frac{2^{2n+1}(n!)^4}{(2n+1)((2n)!)^3} f^{2n}(\xi) \qquad (3\text{-}52)$$

where

$$-1 < \xi < 1$$

The Gaussian quadrature formulas integrate polynomials of twice the degree of the corresponding Newton-Cotes. However, the fact that Gaussian formulas require irrational abscissas limits the use of this method. Our effort on the Gaussian quadrature has been centered on the two-point rule to ten-point rule (see Figure 3-2). The corresponding rounding error coefficients are calculated and are listed as below.

Table 3-4. Rounding error coefficients for Gauss quadrature.

Scheme	Rounding Error Coefficients
Gauss 2 points	5.000
Gauss 3 points	5.556
Gauss 4 points	6.000
Gauss 5 points	6.431
Gauss 6 points	6.532
Gauss 7 points	6.991
Gauss 8 points	6.961
Gauss 9 points	7.454
Gauss 10 points	7.356

The rounding error coefficients for Gauss 8 point and Gauss 10 point rules are each smaller than the rule of a degree lesser. This is caused by multiplying their corresponding weights.

The rounding error for each scheme is bounded by the corresponding coefficient multiplied by δ

$$\delta = (B-A)(1.1)b^{1-t}M$$

H. Repeated Gauss 5 Point Rule (RTGAUSS5)

We extend the Gauss 5 point rule by cutting the interval in half and applying the same formula twice, such as when N is $1, 2, 3,$ or 4, then the number of functions evaluated is 5, 10, 20, or 40 (see Figure 3-2). The rounding error coefficients for this method are governed by the following equations

For $N = 1$

$$TE = 6.431$$

For $N \geq 2$

$$E1 = N + 5 + 2^{(N-1)} + (N+5+2^{(N-1)}+N+7)(N-1)/2$$

$$E2 = N + 3 + 2^{(N-1)} + (N+3+2^{(N-1)}+N+5)(N-1)/2$$

$$TE = (R1*E1*2+R2*E1*2+R3*E2)/(2*2^{(N-1)}) \qquad (3\text{-}53)$$

$$N = 2, 3, 4\ldots$$

where

$$R1 = 0.237, \quad R2 = 0.479, \quad \text{and} \quad R3 = 0.569.$$

The rounding error coefficient for applying repeated Gauss 5 point rule twice is 8.435. It is a little higher than the rounding error coefficient for Gauss 10 point rule, which is 7.356. The rounding error for repeated Gauss 5 point rule is

$$|E| \leq (B-A)*(1.1)*b^{1-t}*M*|TE|$$

3.3. Comparison of Quadrature's Rounding Error

In analyzing the rounding error for each scheme, the error that is introduced by the calculation of abscissas and the function evaluation are not considered. Error introduced by function evaluation is impossible to generalize, as it depends on individual function. Since error introduced by function evaluation is not considered, it is

pointless to evaluate the correct calculation of abscissa, though it is possible. Nevertheless, in many of the schemes, the value of (h) or the interval size has to be measured. The error introduced by computing the interval size then is scruntinized. The rounding error coefficients for each scheme are plotted against the number of function evaluations, except for Gauss and Simpson's adaptive method. For Gauss please refer to Table 3-4. For Simpson's adaptive method, the rounding error can only be specified within a certain range as the scheme can branch out at various stage(s) and various intervals. The best way is to run through the program with the specific integrand. Nevertheless, the rounding error for Simpson's adaptive method is usually higher than other scheme and is comparable to Romberg's. From Figure 3-3, we notice that the Romberg accumulates rounding errors much more than the rest of the schemes. The rounding error for the mid-point rule and the trapezoidal rule are the lowest up to four hundred accumulated function evaluations. For any higher accuracy desired with more function evaluation, Simpson's rule is the one to use for the least effect of rounding errors.

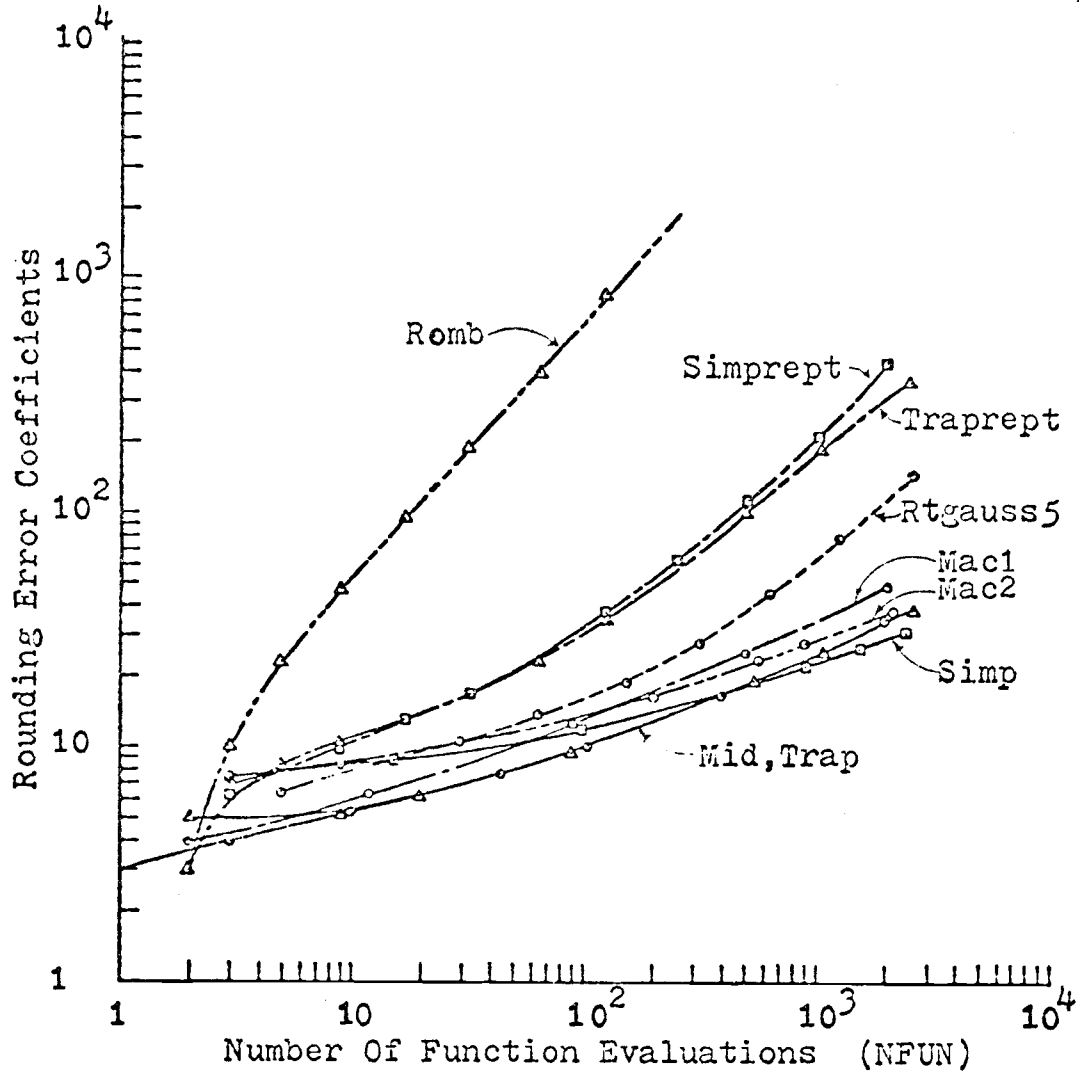


Figure 3-3. Rounding error coefficients versus number of function evaluations.

IV. APPLICATION OF QUADRATURES ON CDC 3300 COMPUTER

4.1. Introduction

In this chapter application of the above analysis to the CDC 3300 is performed. Three properties are investigated, the rounding error, the core storage memory and CPU time.

The CDC 3300 computer has word length of 24 bits/word. A floating point number consists of two words, that is, it has 48 bits altogether. This is divided into segments of the first bit as the sign bit, the next 11 bits as exponent, and the remainder 36 bits as mantissa. Consequently, only 11 decimal digits after the decimal place are available. Double precision for 3300 is available only in addition, subtraction, multiplication and division. There is no double precision of any function evaluation such as trigonometry functions, exponential, square root etc. This limitation restricted our effort to single precision. CPU time analysis, memory storage and rounding error of each scheme are examined.

4.2. Memory

The core storage is analyzed by counting the word length of the program. In the CDC 3300 computer, each page of memory is assigned 2,048 memory locations. A fully expanded system contains 128 of these pages. Programs may be allocated full pages.

In analyzing the programs, Simpson's repeated method takes the least number (134) of locations. This is because it utilizes all the previous function evaluations and does not require storage to keep track of function evaluations. The program for Gaussian formulas from 2-point to 10-point does occupy the maximum amount of memory storage, as more than half of the total abscissas and weights, and function evaluations have to be stored.

Romberg's method required moderate memory storage. The Simpson, trapezoidal and the mid-point method demand storage when more than one thousand function evaluations is needed to achieve the required accuracy. The memory storage for the methods (except Simpson's adaptive) are plotted against function evaluations (NFUN) for comparison on Figure 4-1.

The Simpson's adaptive method uses the most memory storage space. The memory storage for the method depends not only on the number of function evaluations but also on the number of levels that it has to pyramid. The amount of memory storage space required for the method is plotted against level on Figure 4-2.

4.3. Time

In the time analysis of each scheme, an assembly listing of the Fortran program of the scheme is acquired first. The steps for performing input and output in the program (I/0) are not included in the

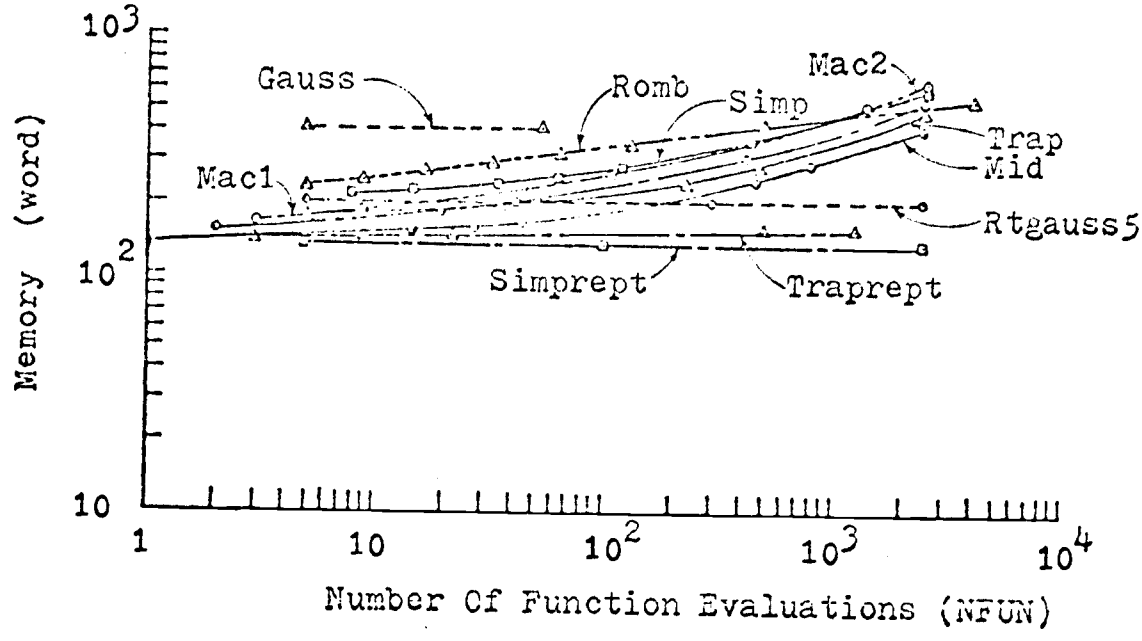


Figure 4-1. Memory requirements of ten numerical quadrature.

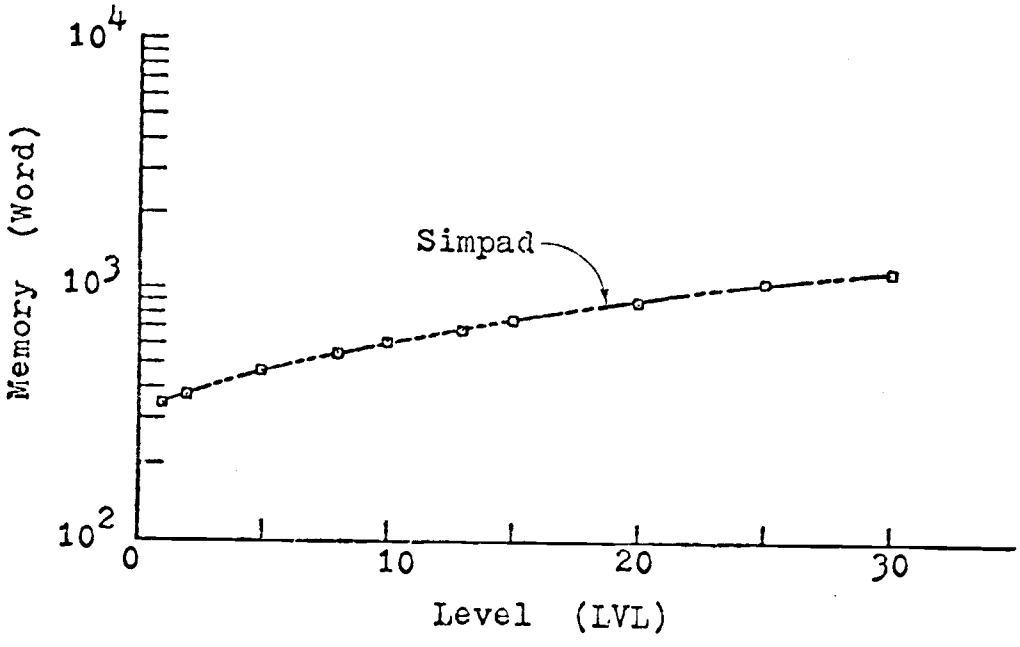


Figure 4-2. Memory requirements of Simpson's adaptive rule.

analysis. The programs from the system library such as the absolute value function, and converting number for mix-mode operation are also considered when necessary. The basic execution time of the machine language (Compass) of the program plus the time for indexing, the time for indirect assessing, and the read next instruction time are compiled. It is assumed that each program can be stored on one page which is in fact found to be the case as indicated in Section 4-2. The re-location time and swapping time are not calculated, as this is considered part of the over-head system time and is not part of the essential time required by the scheme.

The time required for different schemes, except Simpson's adaptive, are plotted against function evaluations (NFUN) on Figure 4-3. These curves are enclosed within the evelope of two straight lines, both with slope 1 and base points at .1 milli-second (ms) and .3 ms. From this we conclude that for a rough estimation, if a would-be user knows the time required for each function evaluation (T1) of the integrand he desires, and the number of function evaluations (N1) the approximation takes to achieve the accuracy, if he multiplies this number of function evaluations by .3 ms and the time required for each function evaluation, this will give him a close approximation of the computer time (T) required.

$$T = N1*(T1+.3)ms \qquad (4-1)$$

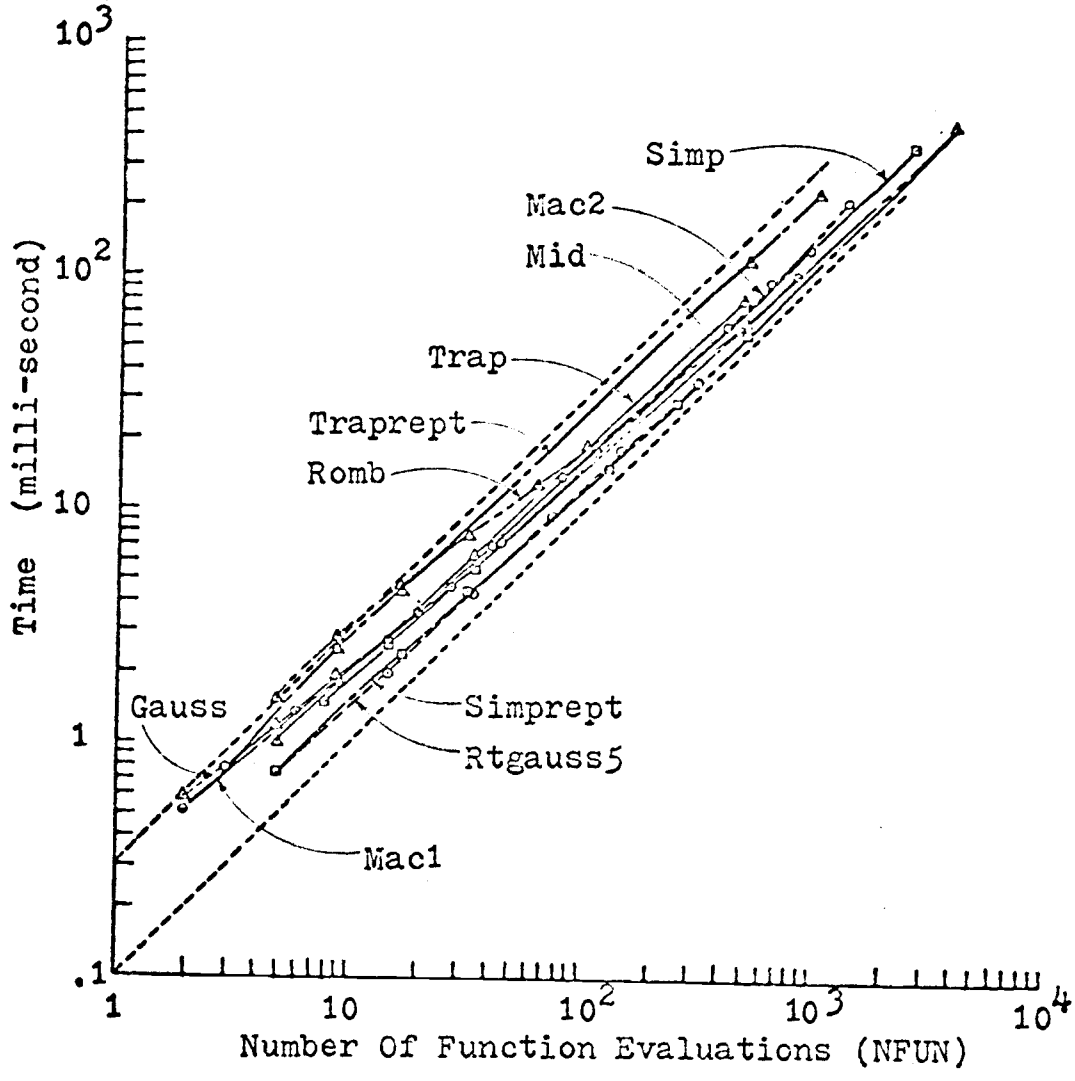


Figure 4-3. Central processing time plotted against number of function evaluations for ten numerical quadrature.

If the number of function evaluations is more than one thousand, he can substitute .3 ms by .25 ms. This is because the time for the supporting steps within the program are equally divided per function evaluation. This applies to all the schemes including Simpson's adaptive method.

4.4. Tested Functions

Five definite integrals are tested, each a representative of a class.

A smooth function with multiple derivatives exist.

$$\int_1^2 \frac{1}{x^2}\,dx = 0.5 \qquad (4\text{-}2)$$

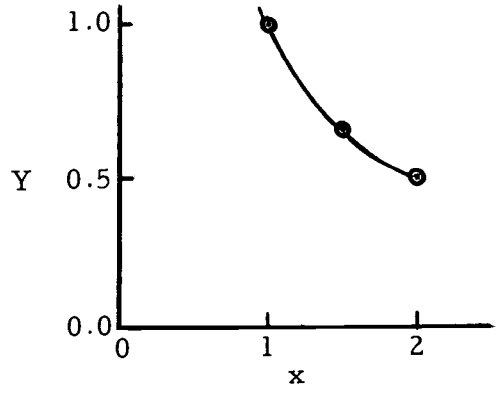


Figure 4-4. $Y = 1/(x^2)$.

A periodic function.

$$\int_0^{2\pi} e^{\sin(x)}\,dx = 7.9549265210133 \qquad (4\text{-}3)$$

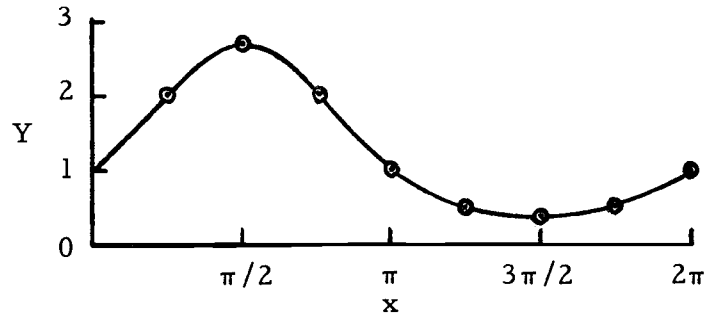


Figure 4-5. $Y = e^{\sin(x)}$.

A function with discontinuity in its first derivative at the center point.

$$\int_{-1}^{1} |x| dx = 1.0 \qquad (4\text{-}4)$$

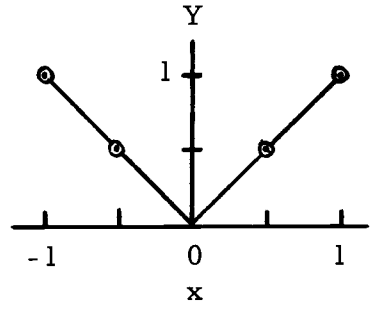


Figure 4-6. $Y = |x|$.

A function with discontinuity in its first derivative at a non-central point.

$$\int_{-1}^{1} |x - \tfrac{1}{7}| dx = 1.020408163265 \qquad (4\text{-}5)$$

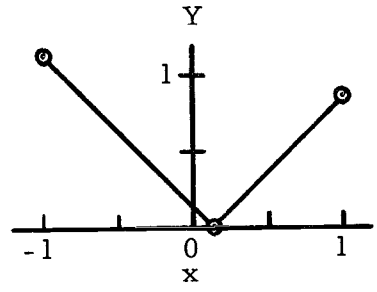


Figure 4-7. $Y = |x-1/7|$.

A function with singularity.

$$\int_0^1 \frac{1}{\sqrt{x}}\,dx = 2.0 \qquad (4\text{-}6)$$

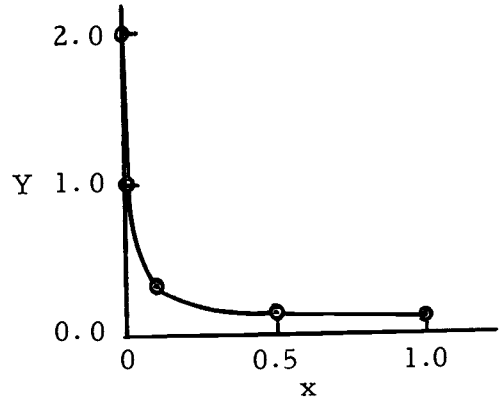


Figure 4-8. $Y = x^{-1/2}$.

Each function is tested for six parameters, i.e., accuracy, N (the index in the programs, see Figure 3-1 and Figure 3-2), NFUN, which is the number of function evaluations, rounding error, memory and time.

For accuracy, 3, 6, and 9 decimal places are the numbers of

significant figures after the decimal point requested in the program. For example, if three decimal place accuracy is expected, an IF statement is used as follows

IF (ABS(SUM - EXACT ANSWER) - 0.001),

where SUM is the computational answer for the integral. This requested accuracy may or may not be satisfied. If the expected accuracy in the program is not satisfied, the corresponding point in the figures (Figure 4-9 to Figure 4-20) is plotted with a hexagon (⊙).

Due to the limited resources of computer time, function evaluations (NFUN) for each scheme is bounded by an IF (NFUN - 2500) statement.

Since we are computing with a binary machine (CDC 3300), the rounding error is calculated by substituting (B-A) by the actual interval, t bits by 36 bits, C_0 by 1, and M, by the highest point of each tested function (except $\int_0^1 \frac{1}{\sqrt{x}}\,dx$, there is no highest point in this case, and the rounding error for this function is not considered). Memory and time are tabulated also in Tables 4-1 to 4-4.

4.5. Results for the Tested Functions

The rounding errors for each of the example computed are applicable only to binary machine with 36 bits of mantissa. The general formula of the rounding error for a computer with base b and

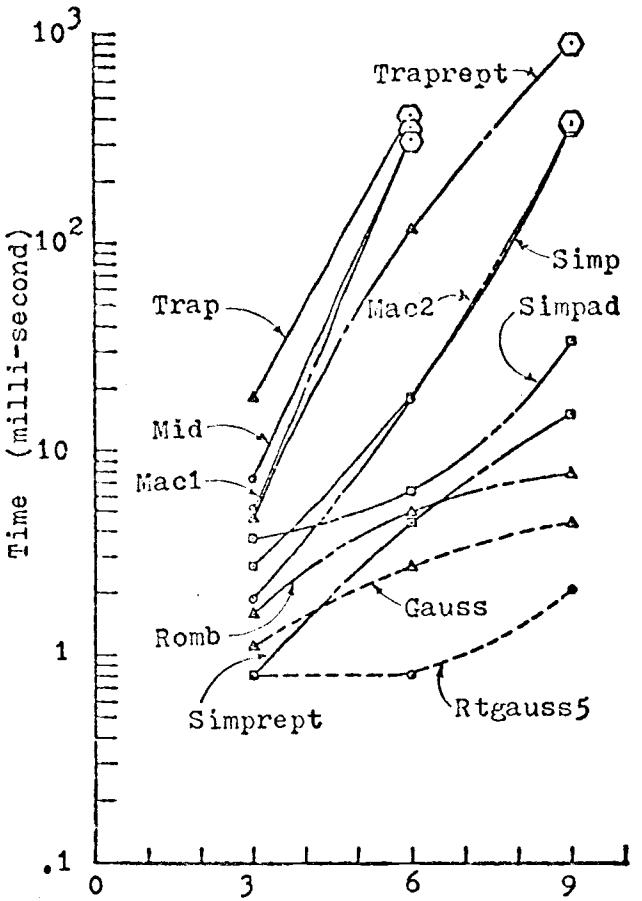


Figure 4-9. Time versus accuracy expected for $\int_1^2 \frac{1}{x^2} dx$.

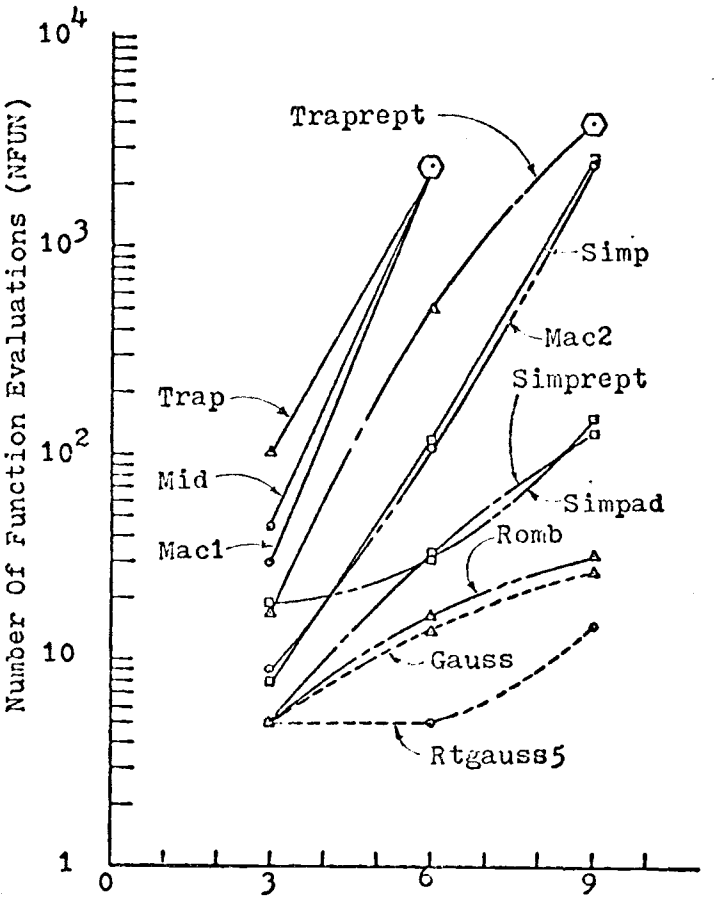


Figure 4-10. NFUN versus accuracy expected for $\int_1^2 \frac{1}{x^2} dx$.

Table 4-1. Table for $\int_1^2 \frac{1}{x^2}\,dx$.

	SUM	N	NFUN	ROUNDING ERROR (*10**11)	MEMORY (WORD)	TIME (MS)
MID	.4991	9	45	25	167	7.2
MAC1	.4993	5	30	27	193	5.3
MAC2	.4998	2	9	27	189	1.9
TRAP	.5009	13	104	32	218	18.2
TRAPREPT	.5006	5	17	41	146	4.6
SIMP	.5004	2	8	27	216	2.7
SIMPREPT	.5004	2	5	26	134	.8
SIMPAD	.5000	*2	19	41	376	3.7
ROMB	.5001	(3,1)	5	74	236	1.6
GAUSS	.4999	1	5	21	417	1.1
RTGAUSS5	.5000	1	5	21	202	.8
3 DECIMAL PLACE ACCURACY EXPECTED						
MID	.4999955	71	2556	125	415	313.5
MAC1	.4999927	50	2550	171	553	363.2
MAC2	.4999992	8	108	45	311	17.9
TRAP	.5001298	70	2555	123	446	412.5
TRAPREPT	.5000006	10	513	323	156	117.1
SIMP	.5000008	10	120	41	280	18.5
SIMPREPT	.5000001	5	33	55	134	4.4
SIMPAD	.5000003	*3	31	58	405	6.4
ROMB	.5000000	(5,1)	17	303	268	4.9
GAUSS	.4999998	1	14	21	417	2.7
RTGAUSS5	.4999998	1	5	21	202	.8
6 DECIMAL PLACE ACCURACY EXPECTED						
MID	.4999955364	71	2556	125	415	313.5
MAC1	.4999927386	50	2550	171	553	363.2
MAC2	.4999999989	41	2583	137	657	387.3
TRAP	.5000029766	70	2555	123	446	412.5
TRAPREPT	.5000000097	13	4097	2245	162	926.0
SIMP	.5000000013	50	2600	105	600	371.0
SIMPREPT	.5000000005	7	129	122	134	15.2
SIMPAD	.5000000003	*4	151	223	434	33.9
ROMB	.5000000000	(6,1)	33	607	290	7.9
GAUSS	.4999999998	1	27	22	417	4.4
RTGAUSS5	.4999999994	2	15	27	202	2.1
9 DECIMAL PLACE ACCURACY EXPECTED						

Table 4-2. Table for $\int_0^{2\pi} \epsilon^{\sin(x)} dx$.

	SUM	N	NFUN	ROUNDING ERROR (*10**11)	MEMORY (WORD)	TIME (MS)
MID	7.9546	3	6	255	143	1.4
MAC1	7.9549	4	20	403	185	3.7
MAC2	7.9551	3	18	515	201	3.5
TRAP	7.9546	3	9	292	178	1.9
TRAPREPT	7.9549	4	9	567	144	2.6
SIMP	7.9546	3	15	498	224	2.7
SIMPREPT	7.9549	4	17	709	134	2.5
SIMPAD	7.9549	*2	19	699	376	3.7
ROMB	7.9549	(6,1)	33	10370	290	7.9
GAUSS	--	-	-	-	-	-
RTGAUSS5	7.9549	4	35	584	202	4.5
3 DECIMAL PLACE ACCURACY EXPECTED						
MID	7.9549265	5	15	317	151	2.9
MAC1	7.9549265	5	30	459	193	5.3
MAC2	7.9549265	5	45	515	225	7.9
TRAP	7.9549265	5	20	339	186	3.9
TRAPREPT	7.9549265	5	17	707	146	4.6
SIMP	7.9549265	5	35	558	240	5.8
SIMPREPT	7.9549265	5	33	941	134	4.4
SIMPAD	7.9549265	*2	19	699	376	3.7
ROMB	7.9549266	(7,1)	65	21362	316	12.9
GAUSS	--	-	-	-	-	-
RTGAUSS5	7.9549265	8	75	755	202	9.2
6 DECIMAL PLACE ACCURACY EXPECTED						
MID	7.9549265208	7	28	375	159	4.8
MAC1	7.9549265206	6	42	515	201	7.2
MAC2	7.9549265213	5	45	515	225	7.9
TRAP	7.9549265206	7	35	391	194	6.4
TRAPREPT	7.9549265211	5	17	707	146	4.6
SIMP	7.9549265209	7	63	615	260	10.1
SIMPREPT	7.9549265212	5	33	941	134	4.4
SIMPAD	7.9549265209	*2	19	699	376	3.7
ROMB	7.9549265212	(8,1)	129	46212	346	21.5
GAUSS	--	-	-	-	-	-
RTGAUSS5	7.9549265211	8	75	755	202	9.2
9 DECIMAL PLACE ACCURACY EXPECTED						

Table 4-3. Table for $\int_{-1}^{1} |x| dx$.

	SUM	N	NFUN	ROUNDING ERROR (*10**11)	MEMORY (WORD)	TIME (MS)
MID	1.0000	2	3	26	139	.8
MAC1	1.0000	1	2	26	161	.6
MAC2	1.0000	1	3	46	177	.8
TRAP	1.0000	2	5	32	174	1.1
TRAPREPT	1.0000	2	3	38	140	.8
SIMP	1.0000	2	8	54	208	1.6
SIMPREPT	1.0000	2	5	52	134	.8
SIMPAD	0.9999	*4	43	149	434	9.1
ROMB	0.9996	(4,1)	9	304	250	2.9
GAUSS	--	-	-	-	-	-
RTGAUSS5	1.0000	2	15	54	202	2.1
3 DECIMAL PLACE ACCURACY EXPECTED						
MID	1.0000000	2	3	26	139	.8
MAC1	1.0000000	1	2	26	161	.6
MAC2	1.0000000	1	3	46	177	.8
TRAP	1.0000000	2	5	32	174	1.1
TRAPREPT	1.0000000	2	3	38	140	.8
SIMP	1.0000000	2	8	54	208	1.6
SIMPREPT	1.0000000	2	5	52	134	.8
SIMPAD	0.9999994	*6	67	213	492	14.7
ROMB	1.0000000	(6,1)	33	1214	290	7.9
GAUSS	--	-	-	-	-	-
RTGAUSS5	1.0000000	2	15	54	202	2.1
6 DECIMAL PLACE ACCURACY EXPECTED						
MID	1.0000000000	2	3	26	139	.8
MAC1	1.0000000000	1	2	26	161	.6
MAC2	1.0000000000	1	3	46	177	.8
TRAP	1.0000000000	2	5	32	174	1.1
TRAPREPT	1.0000000000	2	3	38	140	.8
SIMP	1.0000000000	2	8	54	208	1.6
SIMPREPT	1.0000000000	2	5	52	134	.8
SIMPAD	0.9999999991	*9	103	309	579	22.9
ROMB	1.0000000000	(7,1)	65	2502	316	12.9
GAUSS	--	-	-	-	-	-
RTGAUSS5	1.0000000000	2	15	54	202	2.1
9 DECIMAL PLACE ACCURACY EXPECTED						

Table 4-4. Table for $\int_{-1}^{1} |x-1/7|dx$.

	SUM	N	NFUN	ROUNDING ERROR (*10**11)	MEMORY (WORD)	TIME (MS)
MID	1.0204	7	28	51	159	4.8
MAC1	1.0204	7	56	76	209	9.4
MAC2	1.0204	7	84	95	207	14.1
TRAP	1.0204	7	35	52	194	6.4
TRAPREPT	1.0204	10	513	738	156	117.1
SIMP	1.0204	7	63	82	256	10.1
SIMPREPT	1.0204	8	257	468	134	29.3
SIMPAD	1.0204	*5	55	207	463	11.9
ROMB	1.0204	(7,1)	65	2859	316	12.9
GAUSS	--	-	-	-	-	-
RTGAUSS5	1.0207	4	35	78	202	4.5
3 DECIMAL PLACE ACCURACY EXPECTED						
MID	1.0204082	7	28	51	159	4.8
MAC1	1.0204082	7	56	76	209	9.4
MAC2	1.0204082	7	84	95	207	14.1
TRAP	1.0204082	7	35	52	194	6.4
TRAPREPT	1.0204087	11	1025	1370	158	232.9
SIMP	1.0204082	7	63	82	256	10.1
SIMPREPT	1.0204090	9	513	839	134	57.3
SIMPAD	1.0204082	*7	79	280	521	17.4
ROMB	1.0204084	(10,1)	513	35004	418	65.5
GAUSS	--	-	-	-	-	-
RTGAUSS5	1.0204087	128	1275	570	202	147.7
6 DECIMAL PLACE ACCURACY EXPECTED						
MID	1.0204081633	7	28	51	159	4.8
MAC1	1.0204081633	7	56	76	209	9.4
MAC2	1.0204081633	7	84	95	207	14.1
TRAP	1.0204081633	7	35	52	194	6.4
TRAPREPT	1.0204082206	13	4097	5131	162	926.0
SIMP	1.0204081633	7	63	82	256	10.1
SIMPREPT	1.0204081749	12	4097	5961	134	447.2
SIMPAD	1.0204081633	*10	115	390	608	25.7
ROMB	1.0204083340	(13,1)	4097	707267	556	454.1
GAUSS	--	-	-	-	-	-
RTGAUSS5	1.0204082391	256	2555	1046	202	295.2
9 DECIMAL PLACE ACCURACY EXPECTED						

t digits mantissa is

$$E \leq (B-A)*(1.1)*b^{1-t}*M*(\text{Corresponding rounding error coefficient}) \tag{4-7}$$

If a binary machine with only 15 bits of mantissa is available, then the rounding error can be computed either by the formula

$$E \leq (B-A)*(1.1)*2^{1-15}*M*(\text{Corresponding rounding error coefficient}) \tag{4-8}$$

or by converting our results, such as following.

Let E_1, which is bounded by $2^{1-36}*C$, C being (B-A)*(1.1)*M*(Corresponding rounding error coefficient), be the result of rounding error of one of our example, and E_2, which is bounded by $2^{1-15}*C$, be the corresponding rounding error in the binary machine with 15 bits of mantissa. Then the relation between E_1 and E_2 is

$$E_2 = E_1 * \frac{2^{1-15}}{2^{1-36}} \tag{4-9}$$

For example, the rounding error (E_1) of the mid-point rule for the periodic function $\int_0^{2\pi} e^{\sin(x)} dx$, is bounded by $255*10^{-11}$. This limitation would hardly affect the accuracy of the approximation (7.9546) of the integral (requesting 3 decimal place of expected accuracy and with 36 bits of mantissa available). However, if a

computer, which is designed to have only 15 bits of mantissa, is utilized, the rounding error for solving the same problem

$$E_2 \leq 255*10^{-11} * \frac{2^{-14}}{2^{-35}}$$

$$= 255*2097152*10^{-11}$$

$$= 255*2.1*10^{-5}$$

$$= 5.4*10^{-3} \qquad (4\text{-}10)$$

This indicates that it may affect the result (7.9546). Therefore it can be concluded that with a binary machine of 15 bits of mantissa, to integrate the periodic function $\int_0^{2\pi} e^{\sin(x)} dx$ with mid-point rule, only 2 decimal places of accuracy can be guaranteed.

Now we will proceed to analyze each example in more detail.

A. Smooth Function $\int_1^2 \frac{1}{x^2} dx$

For smooth functions like this, the Gaussian formulas predominate. Repeated Gauss-5 uses the least number of function evaluations and computer time. The rounding error is the lowest with the exception of the Gaussian 7 points. The memory required can be stored within a quarter of a page. For higher accuracy (9 decimal places) the core storage requirements is higher than Simpson repeated rule, but it is less than the others tested.

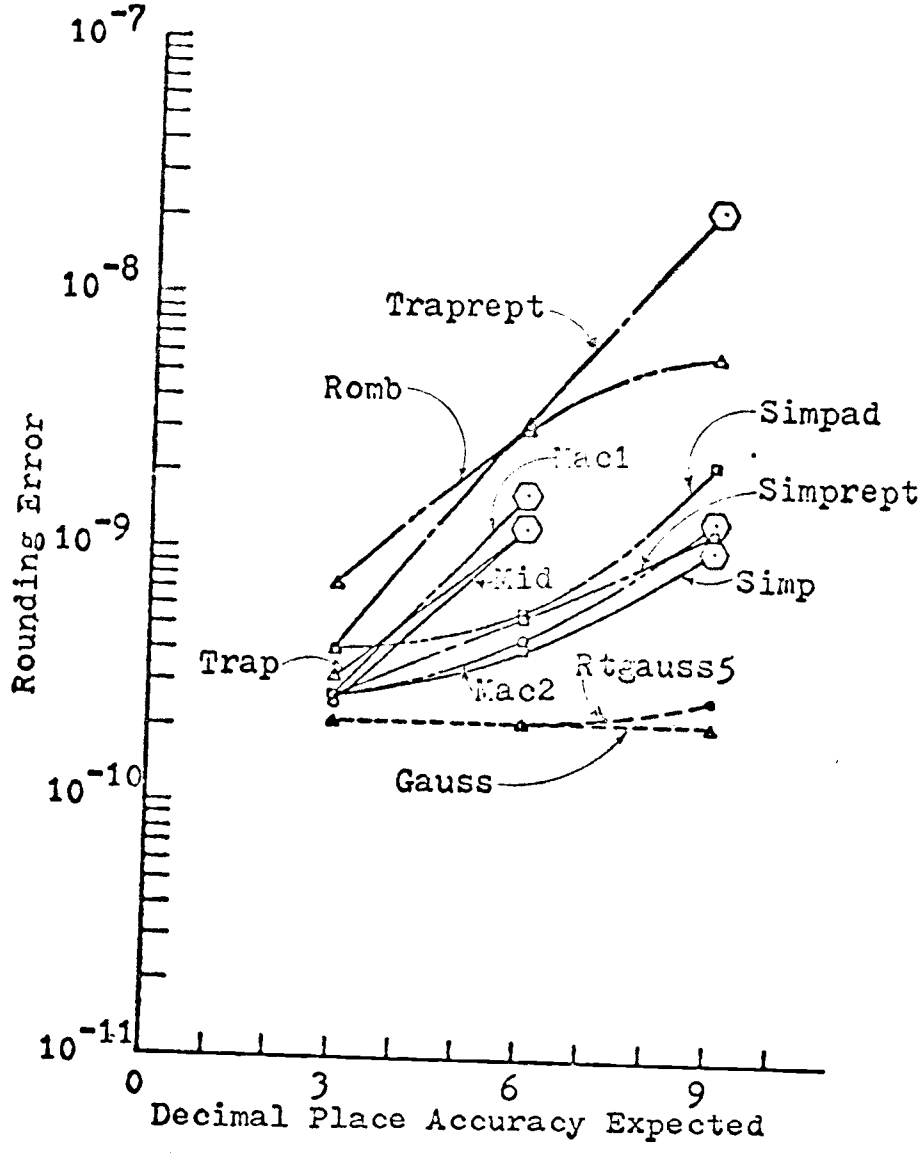


Figure 4-11. Rounding error versus accuracy expected for $\int_1^2 \frac{1}{x^2}\,dx$.

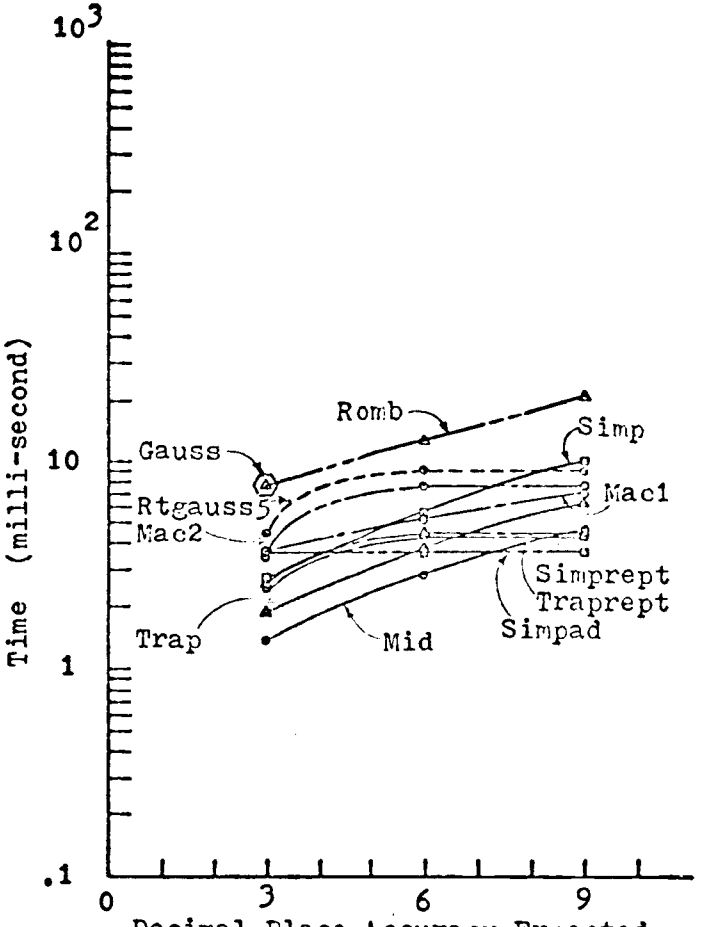


Figure 4-12. Time versus accuracy expected for $\int_0^{2\pi} e^{\sin(x)} dx$.

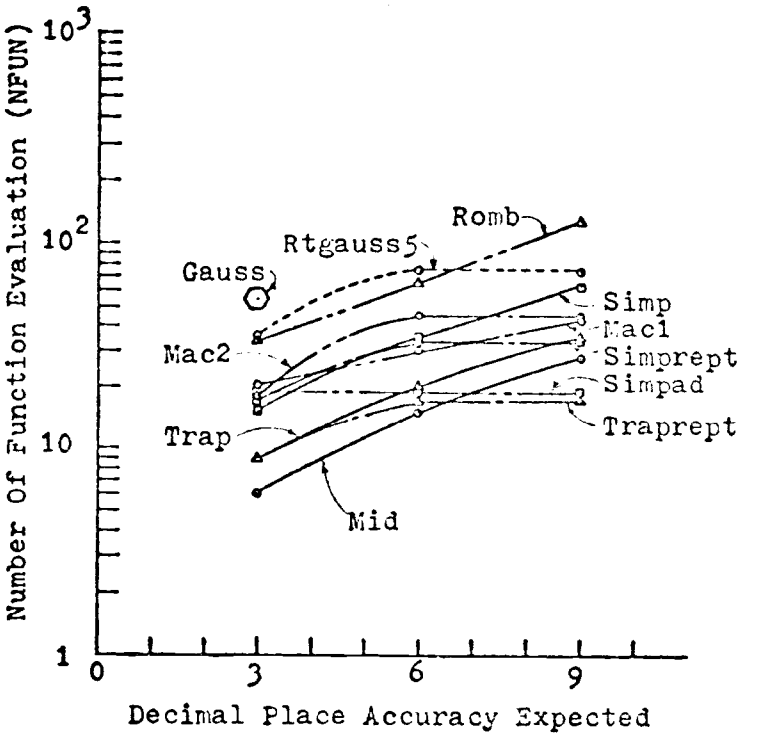


Figure 4-13. NFUN versus accuracy expected for $\int_0^{2\pi} e^{\sin(x)} dx$.

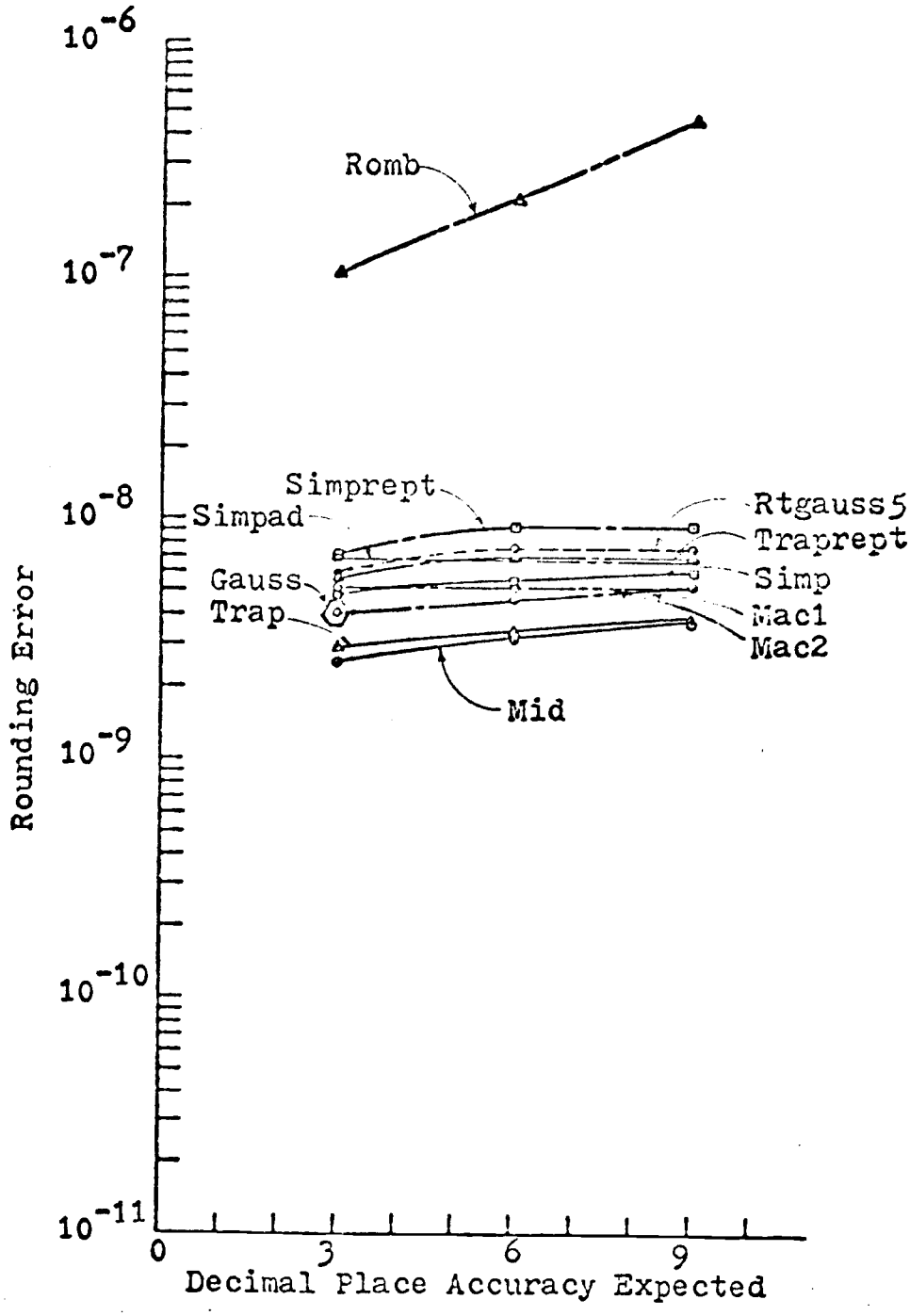


Figure 4-14. Rounding error versus accuracy expected for $\int_0^{2\pi} e^{\sin(x)} dx$.

B. Periodic Function $\int_0^{2\pi} e^{\sin(x)} dx$

The Mid-point rule requires the least number of function evaluations and computer time up to 6 decimal places of accuracy. For a 9 decimal place accuracy, Simpson's Adaptive method requires a smaller number of function evaluations and less computer time. However the mid-point rule still has the smallest rounding error of all the methods and demands memory requirement only more than Simpson and trapezoidal repeated methods.

C. Function with Central Discontinuity in First Derivative $\int_{-1}^{1} |x| dx$

The corner point of this function is at the center. Mid-point rules dominate here, especially the two points mid-point rule (MAC1) (the first rule of the Maclaurin method). It takes the least amount of computer time, has the least number of function evaluations and rounding error. Only the mid-point rule and Simpson repeated rule have less memory storage requirements.

D. Function with Shifted Discontinuity in First Derivative $\int_{-1}^{1} |x-1/7| dx$

For this function the Mid-point rule excells in every respect except for memory requirements in which it ranks only higher than the Simpson's repeated method.

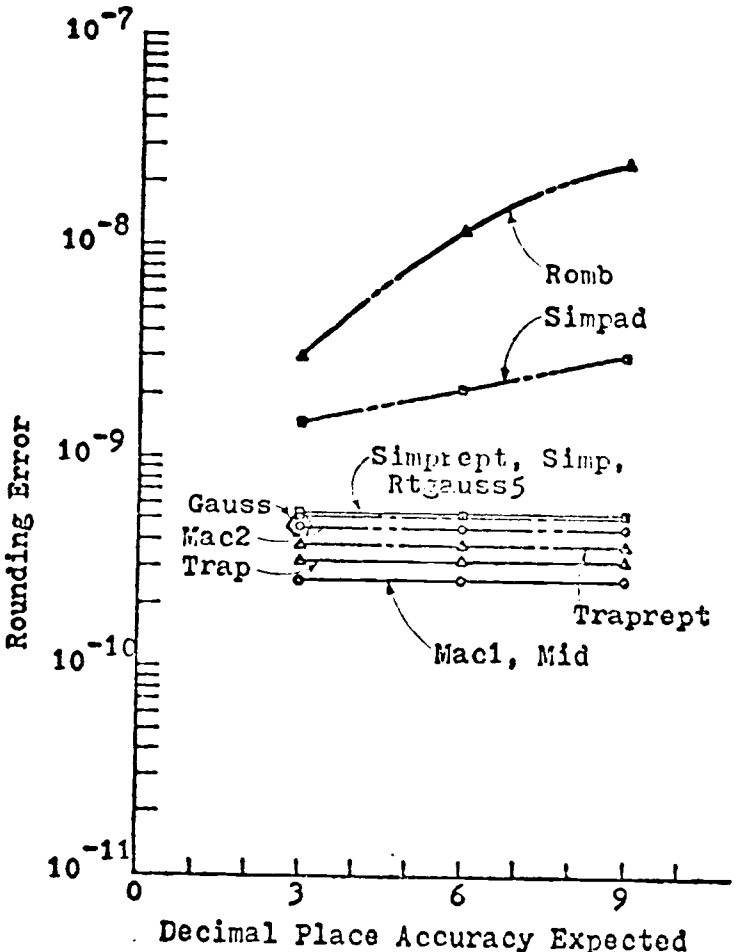


Figure 4-15. Rounding error versus accuracy expected for $\int_{-1}^{1} |x| dx$.

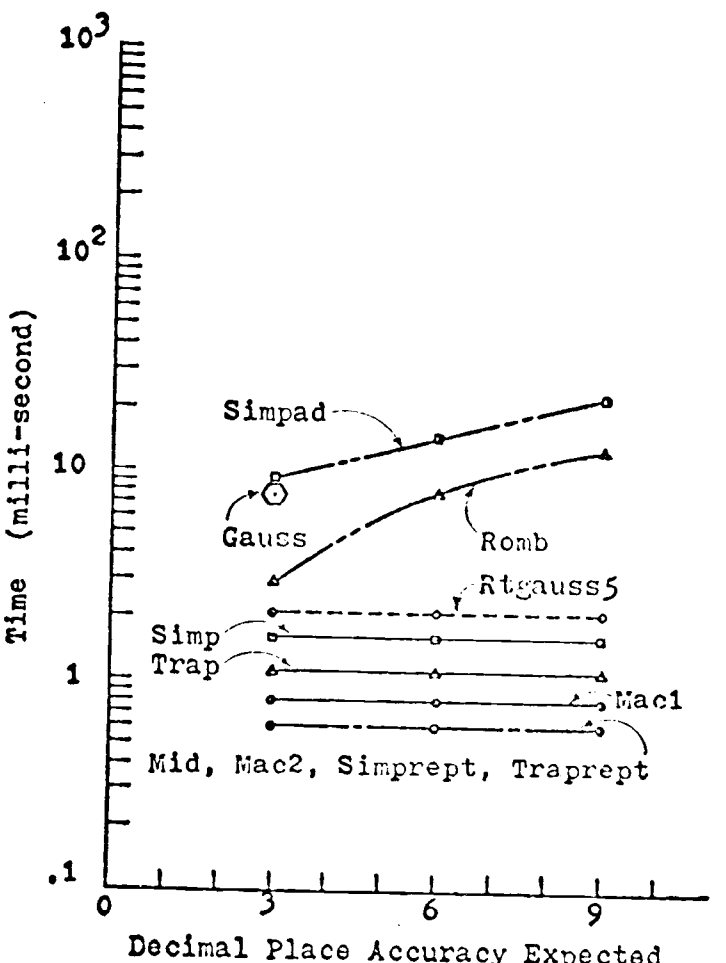


Figure 4-16. Time versus accuracy expected for $\int_{-1}^{1} |x| dx$.

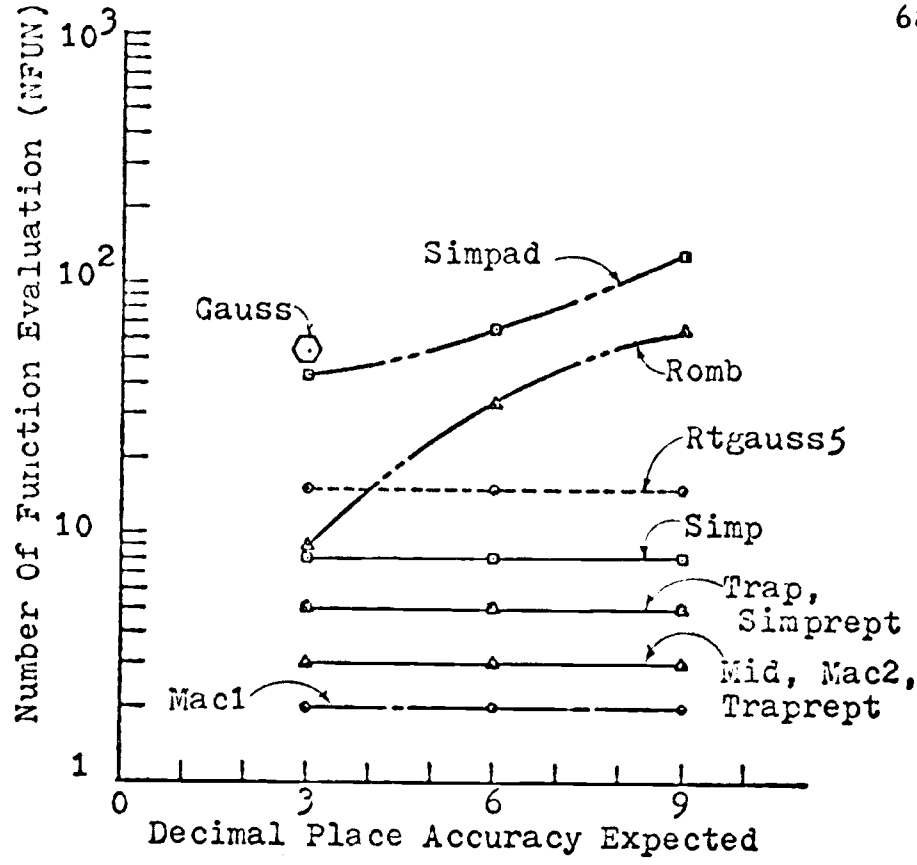


Figure 4-17. Number of function evaluation versus accuracy expected for $\int_{-1}^{1} |x| dx$.

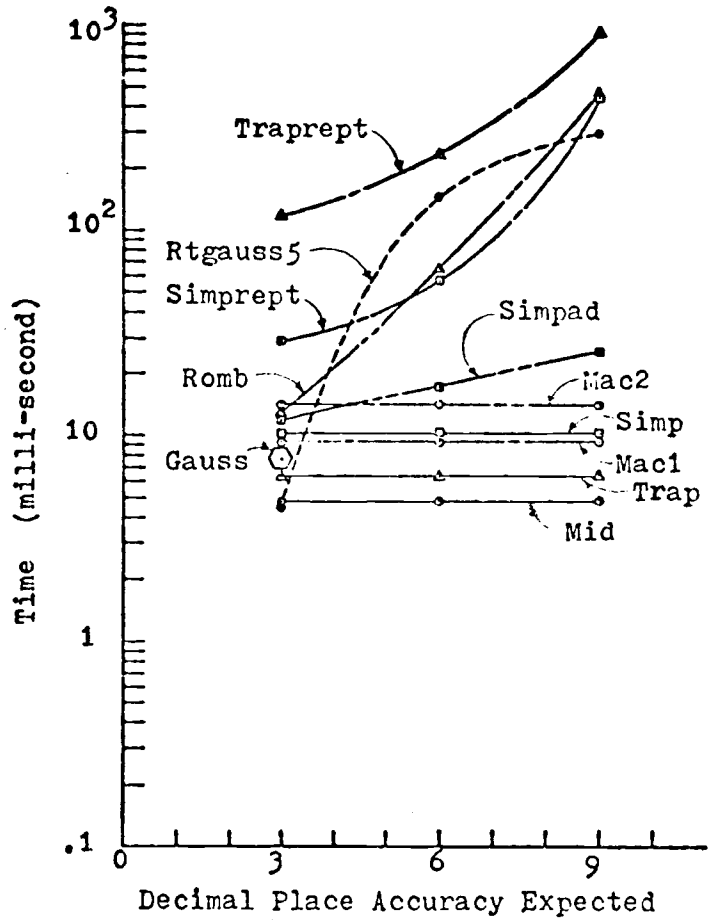


Figure 4-18. Time versus accuracy expected for $\int_{-1}^{1} |x-1/7|dx$.

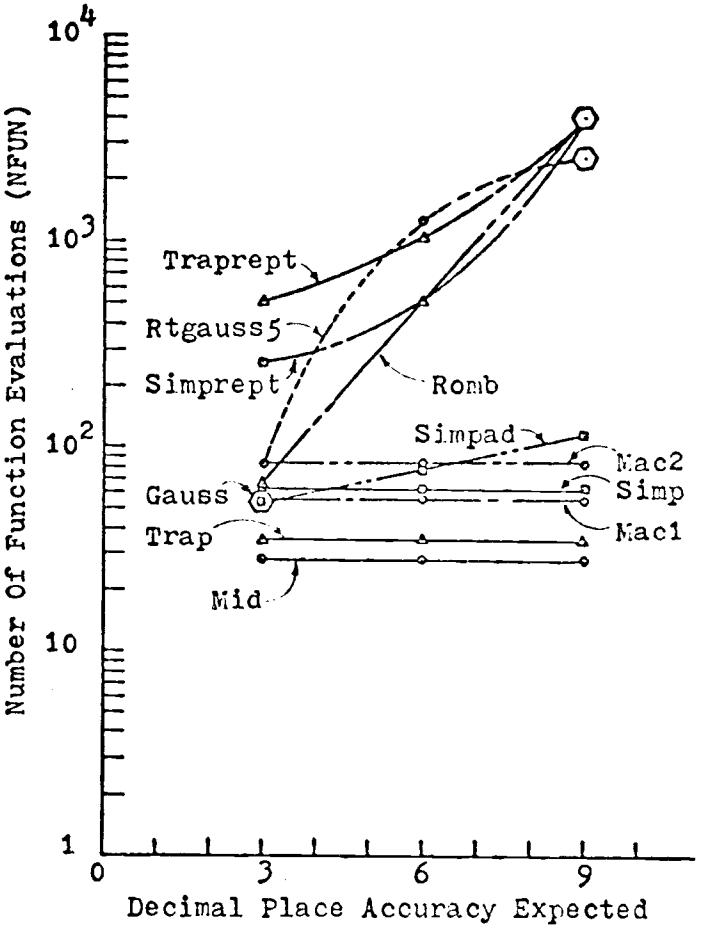


Figure 4-19. NFUN versus accuracy expected for $\int_{-1}^{1} |x-1/7|dx$.

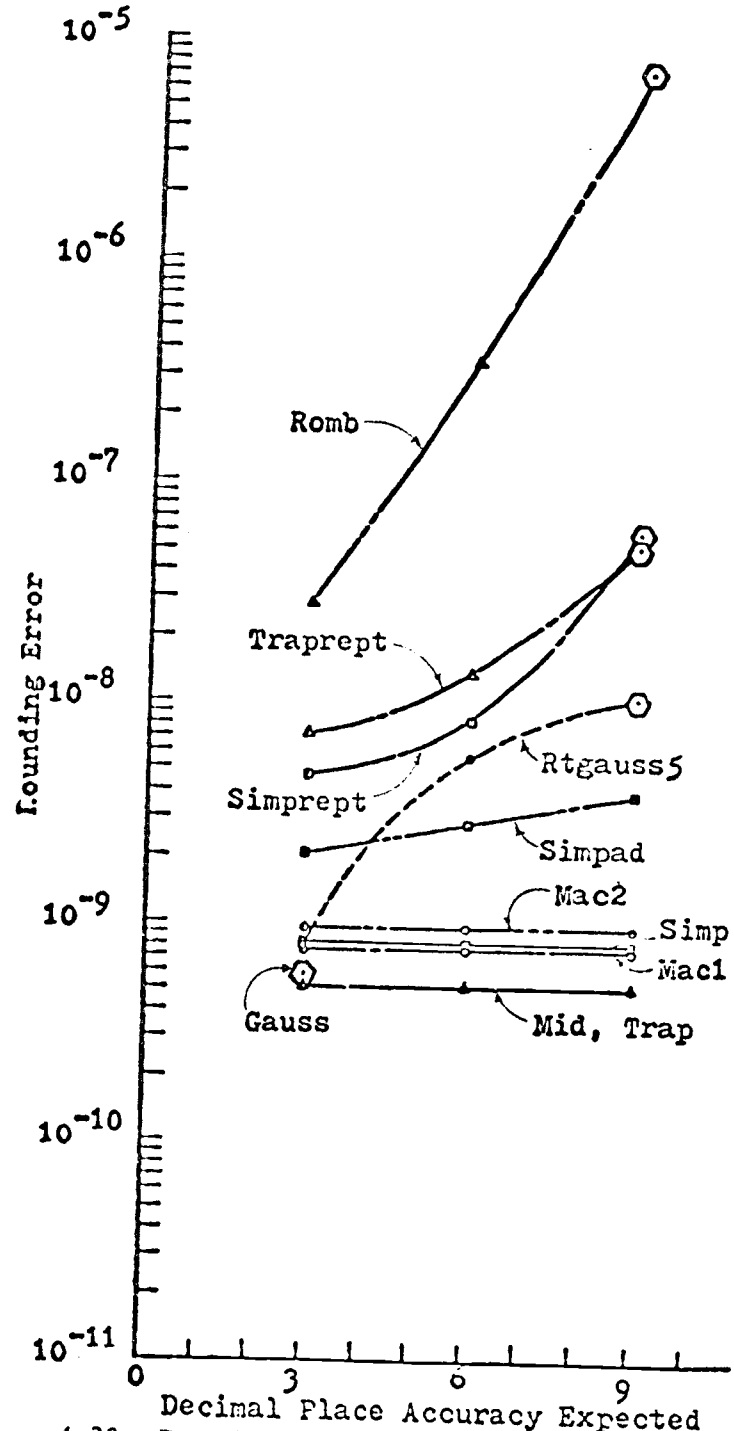


Figure 4-20. Rounding error versus accuracy expected for $\int_{-1}^{1} |x-1/7|dx$.

E. Function with a Singularity $\int_0^1 \frac{1}{\sqrt{x}}\,dx$

A function with a singularity is a disaster for regular mechanical quadratures. Here, Simpson's adaptive method demonstrates its effectiveness. The result is listed as following:

SUM	LVL	NFUN	MEMORY (WORD)	TIME (MS)
2.0000	30	354	1188	80.7
3 decimal place accuracy expected				
2.0000008	30	690	1188	157.7
6 decimal place accuracy expected				
1.999999984	30	4050	1188	927.7
9 decimal place accuracy expected				

Rounding error analysis is not applicable in here (see Section 4-4)

However, when confronting integrands with singular points, a little thinking is often effective. One can usually apply different methods to simplify the approximation such as isolating the singular point, adding and subtracting an additional term, converting into trigonomentric functions, and last of all by simply discarding the small terms so as to save a significant amount of computer time. For example, if we have to integrate

$$S = \int f(x)dx = \int_0^1 \frac{\cos(x)}{\sqrt{x}}\,dx\,, \qquad (4\text{-}11)$$

which has a singular point at zero, we can modify the problem by adding and subtracting $(1-(x^2/2))/\sqrt{x}$ from the function $f(x)$ as following

$$S = \int_0^1 \left(\frac{\cos(x)}{\sqrt{x}} - \frac{1-\frac{x^2}{2}}{\sqrt{x}}\right)dx + \int_0^1 \frac{1-x^2}{\sqrt{x}}\,dx ,$$

$$= \int_0^1 \frac{\cos(x)-1+\frac{x^2}{2}}{\sqrt{x}}\,dx + \int_0^1 (x^{-1/2}-x^{3/2})dx \qquad (4\text{-}12)$$

However,

$$\int_0^1 (x^{-1/2}-x^{3/2})dx = 1.8$$

and by applying SIMPAD

$$\int_0^1 \left(\frac{\cos(x)-1+\frac{x^2}{2}}{\sqrt{x}}\right)dx = 0.0090484758$$

Thus

$$S = 0.0090484758 + 1.8$$

$$= 1.8090484758$$

However, in applying Simpson's adaptive method to the original problem,

$$\int_0^1 \frac{\cos(x)}{\sqrt{x}}\,dx = 1.8090484138 \qquad (4\text{-}13)$$

seven decimal place of accuracy can be achieved with approximately ten times the CPU time.

4.6. Summaries and Conclusions

The following observations are derived from the above results.

1. In comparing the methods, for the same number of function evaluations, it is summarized that,
   A. Simpson's repeated method (SIMPREPT) requires the least of core space,
   B. mid-point rule and trapezoidal method accumulate the least amount of rounding error up to four hundred function evaluations. For more function evaluations, Simpson's rule has less rounding error, and
   C. the CPU time for numerical quadrature (T) depends on the time for each function evaluation (T1), the number of function evaluations (N1), and a constant α, where α is the upper bound of the average scheme time per function evaluation, which in our case, for the computer CDC 3300, α is .3 ms.

$$T = N1*(T1+\alpha)ms \ . \qquad (4\text{-}14)$$

2. In testing the methods with five different integrands on the computer CDC 3300, the following table is evolved. It is noted in here that the more sophisticated methods such as the Romberg, Simpson's adaptive, and the most commonly

Table 4-5. Results of tested functions.

Tested Functions	Least Number of Function Evaluations	Least Accumulated Rounding Errors	Least* Core Memory Requirements	Least Amount of CPU Time	Desired Accuracy Requested by the Program (Decimal Place)
Smooth $\int_1^2 \frac{1}{x^2}\,dx$	RTGAUSS5	RTGAUSS5 and GAUSS	SIMPREPT	RTGAUSS5	3 or 6 or 9
Periodic $\int_0^{2\pi} e^{\sin(x)}dx$	MID	MID	SIMPREPT	MID	3 or 6
	SIMPAD	MID	SIMPREPT	SIMPAD	9
Center discontinuity $\int_{-1}^{1} \lvert x\rvert dx$	MAC1	MID and MAC1	SIMPREPT	MAC1	3 or 6 or 9
Distorted discontinuity $\int_{-1}^{1} \lvert x-1/7\rvert dx$	MID	MID	SIMPREPT	MID	3 or 6 or 9
Singularity** $\int_0^1 \frac{1}{\sqrt{x}}\,dx$	SIMPAD	SIMPAD	SIMPREPT	SIMPAD	3 or 6 or 9

* See discussion in Section 4.2.

** See discussion in Section 4.5.E.

used Simpson's rule do not always have much advantage over the mid-point rule.

3. Although the rounding errors do not affect much of the accuracy of the tested functions in our examples, it has been demonstrated that for a smaller word-length, the rounding errors may become more influential.

To conclude, this paper can be used as a guide for those who are left floundering in the sea of numerical integration computing formulas, whose relative merits are far from clear. They have to know the characteristics of the function that they are integrating as well as the time needed for each of the functions to be evaluated. They can compare the characteristics of their integrand with our test functions and choose the method that is most suitable for their purposes and the degree of accuracy required. From this, they can derive the number of function evaluations needed, the core memory that is required, the upper bound of the computational rounding error that may affect the accuracy, and last of all the computational time that would be required on the computer.

BIBLIOGRAPHY

1. Bauer, F.L., H. Rutishauser and E. Stiefel. "New aspects in numerical quadrature." In: Experimental arithmetic, high speed computing, and mathematics. Providence, American Mathematical Society, 1963. p. 199-218.

2. Davis, Philip J. and Philip Rabinowitz. Numerical integration. Massachusetts, Waltham, 1967. 229 p.

3. Hildebrand, F.B. Introduction to numerical analysis. New York, McGraw-Hill, 1956. 511 p.

4. Hill, Herbert S. Advances in numerical quadrature. In: Mathematical methods for digital computers. Vol. II. New York, Wiley, 1967. p. 133-168.

5. Lyness, J.N. Notes on the adaptive Simpson quadrature routine. Journal of the Association of Computer Machinery 16:483-495. 1963.

6. McKeeman, W.M. and L. Tesler. [A1] "Algorithm 182, nonrecursive adaptive integration." Communication of the Association for Computing Machinery 6:315. 1963.

7. Wilkinson, J.H. Rounding errors in algebraic process. New Jersey, Prentice-Hall, 1963. 161 p.

APPENDIX

APPENDIX I

A. Fortran Program for MID

```
      PROGRAM FUNSINX
C     MID POINT RULE *
      DIMENSION X(1000),Y(1000)
      A=1.
      B=2.
      N=1
100   DELTA =(B-A)/N
      SUM=0.0
      C=A+0.5*DELTA
      DO 104 I=1,N
      X(I)=C+(I-1)*DELTA
      CALL FUN(X(I),Y(I))
      SUM=SUM+Y(I)
104   CONTINUE
      SUM=SUM*DELTA
      ERR=SUM-0.50000000
      IF (ABS(ERR)-0.001)130,106,106
106   N=N+1
      GO TO 100
130   CONTINUE
      WRITE(61,10) SUM
10    FORMAT (6H AREA=,E20.10)
      END
```

B. Fortran Program for MAC1

```
      PROGRAM FUNSINX
C     MID POINT RULE 1
      DIMENSION X(1000),Y(1000)
      A=1.
      B=2.
      N=1
100   NN=2*N
      SUM=0.0
      DELTA=(B-A)/NN
      C=A+.5*DELTA
      DO 104 I=1,NN
      X(I)=C+(I-1)*DELTA
104   CALL FUN (X(I),Y(I))
      DO 110 J=1,NN,2
110   SUM=SUM+Y(J)+Y(J+1)
      SUM=SUM*DELTA
      ERR=SUM-0.50000000
      IF (ABS(ERR)-0.001) 130,106,106
106   N=N+1
      GO TO 100
130   CONTINUE
      WRITE (61,10) SUM
10    FORMAT (6H AREA=,E20.10)
      END
```

C. Fortran Program for MAC2

```
      PROGRAM FUNSINX
C     MID POINT RULE 2
      DIMENSION X(1000),Y(1000)
      A=1.
      B=2.
      N=1
100   NN=3*N
      SUM=0.0
      DELTA =(B-A)/NN
      C=A+0.5*DELTA
      DO 104 I=1,NN
      X(I)=C+(I-1)*DELTA
104   CALL FUN(X(I),Y(I))
      DO 110 J=1,NN,3
      SUM=SUM+3.*(Y(J)+Y(J+2))
110   SUM=SUM+2.*Y(J+1)
      SUM=SUM*3.*DELTA/8.
      ERR=SUM-0.50000000
      IF (ABS(ERR)-0.001) 130,106,106
106   N=N+1
      GO TO 100
130   CONTINUE
      WRITE(61,10) SUM
10    FORMAT (6H AREA=,E20.10)
      END
```

D. Fortran Program for TRAP

```
      PROGRAM FUNSINX
C     TRAPEZOIDAL RULE
      DIMENSION X(1000),Y(1000)
      A=1.
      B=2.
      N=1
      SUM=0.0
100   NN=N+1
      DELTA=(B-A)/N
      DO 104 I=1,NN
      X(I)=A+(I-1)*DELTA
104   CALL FUN (X(I),Y(I))
      SUM=(Y(1)+Y(NN))/2.
      NN2=NN-1
      DO 110 J=2,NN2
110   SUM=SUM+Y(J)
      SUM=DELTA*SUM
      ERR=SUM-0.500000000
      IF (ABS(ERR)-0.0001) 130,106,106
106   N=N+1
      GO TO 100
130   CONTINUE
      WRITE(61,10) SUM
10    FORMAT ( ' AREA = ',E20.10)
      END
```

E. Fortran Program for TRAPREPT

```
      PROGRAM FUNPOMB
C     REPEATED TRAPEZOIDAL RULE
      DIMENSION T(100)
      K=0
      I=1
      A=1.
      B=2.
      DELTA=B-A
      CALL FUN(A,F)
      CALL FUN(B,Y)
      T(1)=DELTA*(F+Y)/2.
10    K=K+1
      DELTA=DELTA/2.
      SUM=0.0
      DO 20 J=1,I
      C=2*J-1
      X=A+C*DELTA
      CALL FUN(X,Y)
20    SUM=SUM+Y
      I=2*I
      T(K+1)=T(K)/2.+SUM*DELTA
      DO 600 MP=1,K
      WRITE (61,50)(MP, T(MP))
50    FORMAT (1X,'T(',I3,') = ',E20.10)
600   CONTINUE
      IF (K-20) 10,10,60
60    CONTINUE
      END
```

F. Fortran Program for ROMB

```
      PROGRAM FUNROMB
C     ROMBERG QUADRATURE
      DIMENSION T(100,100)
      K=0
      I=1
      A=1.
      B=2.
      DELTA=B-A
      CALL FUN(A,F)
      CALL FUN(B,Y)
      T(1,1)=DELTA*(F+Y)/2.
10    K=K+1
      DELTA=DELTA/2.
      SUM=0.0
      DO 20 J=1,I
      C=2*J-1
      X=A+C*DELTA
      CALL FUN(X,Y)
20    SUM=SUM+Y
      I=2*I
      T(1,K+1)=T(1,K)/2.+SUM*DELTA
      J1=1
      DO 30 L=1,K
      N=L+1
      M=K+1-L
      J1=4*J1
30    T(N,M)=T(N-1,M+1)+(T(N-1,M+1)-T(N-1,M))/(J1-1)
      EPS=0.0001
      IF (ABS(T(N,M)-T(N-1,M))-EPS)40,40,10
40    CONTINUE
      N1=N+1
      DO 60 NP=1,N
      N1=N1-1
      DO 60 MP=1,N1
      WRITE (61,50)(NP,MP, T(NP,MP))
50    FORMAT (1X,4HT(4,I3,4,4,I3,4) = 4,E20.10)
60    CONTINUE
      END
```

G. Fortran Program for ROMB with Calculation of Rounding Error Coefficients

```
      PROGRAM FUNROMB
C     ROMBERG QUADRATURE
      DIMENSION T(50,50)
      DIMENSION ET3(50,50)
      DIMENSION ET(50,50)
      K=0
      I=1
      A=0.
      B=1.
      DELTA=B-A
      CALL FUN(A,F)
      CALL FUN(B,Y)
      ET(1,1)=2.
      ET3(1,1)=ET(1,1)*2*1.1/2.**(35)
      T(1,1)=DELTA*(F+Y)/2.
10    K=K+1
      DELTA=DELTA/2.
      SUM=0.0
      DO 20 J=1,I
      C=2*J-1
      X=A+C*DELTA
      CALL FUN(X,Y)
20    SUM=SUM+Y
      I=2*I
      ET(1,K+1)=(-11./3.*0.5**K+2.**(K+1)/12.+(K+1)+11./2.)
      ET3(1,K+1)=ET(1,K+1)*2*1.1/2.**(35)
      T(1,K+1)=T(1,K)/2.+SUM*DELTA
      J1=1
      DO 30 L=1,K
      N=L+1
      M=K+1-L
      J1=4*J1
      ET(N,M)=ABS(ET(N-1,M+1))*4./3.+ABS(ET(N-1,M))/3.+1.
      ET3(N,M)=ET(N,M)*2*1.1/2.**(35)
      T(N,M)=T(N-1,M+1)+(T(N-1,M+1)-T(N-1,M))/(J1-1)
      IF (N-15)30,40,40
30    CONTINUE
      EPS=0.0000001
      IF (ABS(T(N,M)-2.0)-EPS)40,40,10
40    CONTINUE
      WRITE (10,53)
      WRITE (10,53)
      NN=10
      DO 100 NP=1,NN
      WRITE (10,52) (NP,MP,ET(NP,MP),MP=1,5)
52    FORMAT (4X,5(2X*(*,I2,*,*,I2,*)=*,E9.2))
100   CONTINUE
      DO 101 NN=11,15
      NP=NN
      MM=16-NN
      WRITE (10,52) (NP,MP,ET(NP,MP),MP=1,MM)
101   CONTINUE
      WRITE (10,53)
53    FORMAT (////)
      MN=5
      DO 102 NP=1,NN
      WRITE (10,52) (NP,MP,ET(NP,MP),MP=6,10)
102   CONTINUE
      DO 103 NN=6,10
      MP=NN
      MM=16-NN
      WRITE (10,52) (NP,MP,ET(NP,MP),MP=6,MM)
103   CONTINUE
      WRITE (10,53)
      DO 104 NN=1,5
      NP=NN
      MM=16-NN
      WRITE (10,52) (NP,MP,ET(NP,MP),MP=11,MM)
104   CONTINUE
      END
```

H. Fortran Program for SIMP

```
      PROGRAM FUNSINX
C     SIMPSON'S RULE
      DIMENSION X(1000),Y(1000)
      A=1.
      B=2.
      N=1
100   NN=2*N+1
      DELTA =(B-A)/(2*N)
      DO 104 I=1,NN
      X(I)=A+(I-1)*DELTA
      CALL FUN(X(I),Y(I))
104   CONTINUE
      NN2=NN-1
      SUM1=Y(1)+Y(NN)
      SUM2=Y(2)
      SUM3=0.
      DO 110 J=4,NN2,2
      SUM2=SUM2+Y(J)
110   SUM3=SUM3+Y(J-1)
      SUM=DELTA*(SUM1+4.*SUM2+2.*SUM3)/3.
      ERR=ABS(SUM-0.500000000)
      IF (ERR-0.00000001) 130,106,106
106   N=N+1
      GO TO 100
  130   CONTINUE
      WRITE(61,10) SUM
10    FORMAT (6H AREA=,E20.10)
      END
```

I. Fortran Program for SIMPREPT

```
      PROGRAM REPSIMP
C     REPEATED SIMPSON RULE
      N=1
      I=1
      A=1.
      B=2.
      DELTA=B-A
      CALL FUN(A,F)
      CALL FUN(B,Y)
      SUM1=F+Y
10    DELTA=DELTA/2.
      SUM2=0.0
      DO 20 J=1,I
      C=2*J-1
      X=A+C*DELTA
      CALL FUN(X,Y)
20    SUM2=SUM2+Y
      I=2*I
      SUM=SUM1+4.*SUM2
      SUM=DELTA*SUM/3.
      ERR=SUM-0.5
      IF (ABS(ERR)-0.001) 130,106,106
106   N=N+1
      SUM1=SUM1+2.*SUM2
      GO TO 10
130   WRITE (61,51) SUM
51    FORMAT ( ' AREA = ',E20.10)
      END
      SUBROUTINE FUN (X,Y)
      Y=1./(X*X)
      END
```

J. Fortran Program for SIMPAD

```
      SUBROUTINE SIMP(A,B,EPS,SUM)
C     NONRECURSIVE ADAPTIVE SIMPSON INTEGRATION
      DIMENSION DX(30),EPSP(30),X2(30),X3(30),Y2(30),Y3(30),
     1Y4(30),YMP(30),YBP(30),EST2(30),EST3(30),PVAL(30,3),NR(30)
C     THE PARAMETER SET UP FOR THE INITIAL CALL
      LVL=0
      ABSAR=0.0
      EST=0.0
      DA=B-A
      CALL FUN(A,YA)
      CC=(A+B)/2.
      CALL FUN(CC,YM)
      CALL FUN(B,YB)
C     1=RECUR
10    LVL=LVL+1
      DX(LVL)=DA/3.
      SX=DX(LVL)/6.
      C1=A+DX(LVL)/2.
      CALL FUN(C1,Y1)
      X2(LVL)=A+DX(LVL)
      CALL FUN (X2(LVL),Y2(LVL))
      X3(LVL)=X2(LVL)+DX(LVL)
      CALL FUN (X3(LVL),Y3(LVL))
      EPSP(LVL)=EPS
      C4=DX(LVL)/2.+X3(LVL)
      CALL FUN(C4,Y4(LVL))
      YMP(LVL)=YM
      YBP(LVL)=YB
      EST1=SX*(YA+4.*Y1+Y2(LVL))
      EST2(LVL)=SX*(Y2(LVL)+Y3(LVL)+4.*YM)
      EST3(LVL)=SX*(Y3(LVL)+4.*Y4(LVL)+YB)
      SUM=EST1+EST2(LVL)+EST3(LVL)
      ABSAR=ABSAR-ABS(EST)+ABS(EST1)+ABS(EST2(LVL))+
     1ABS(EST3(LVL))
      IF(ABS(EST-SUM)-EPSP(LVL)*ABSAR)20,20,30
30    IF(LVL-30)40,20,20
C     2=UP
20    LVL=LVL-1
      L=NR(LVL)
      PVAL(LVL,L)=SUM
      GO TO (11,12,13)L
C     11=R1,12=R2,13=R3
40    NR(LVL)=1
      EST=EST1
      YM=Y1
      YB=Y2(LVL)
70    EPS=EPSP(LVL)/1.7
      DA=DX(LVL)
      GO TO 10
11    NR(LVL)=2
      YA=Y2(LVL)
      YM=YMP(LVL)
      YB=Y3(LVL)
      EST=EST2(LVL)
      A=X2(LVL)
      GO TO 70
12    NR(LVL)=3
      YA=Y3(LVL)
      YM=Y4(LVL)
      YB=YBP(LVL)
      EST=EST3(LVL)
      A=X3(LVL)
      GO TO 70
13    SUM=PVAL(LVL,1)+PVAL(LVL,2)+PVAL(LVL,3)
      IF (LVL-1) 50,50,20
50    CONTINUE
      END
```

K. Fortran Program for SIMPAD with Calculation of Rounding Error Coefficients

```
      SUBROUTINE SIMP(A,B,EPS,SUM)
C     NONRECURSIVE ADAPTIVE SIMPSON INTEGRATION
      DIMENSION DX(30),EPSP(30),X2(30),X3(30),Y2(30),Y3(30),
     1Y4(30),YMP(30),YBP(30),EST2(30),EST3(30),PVAL(30,3),NR(30)

      DIMENSION EPVAL(30,3)
C     THE PARAMETER SET UP FOR THE INITIAL CALL
      ESUM=0.0
      LVL=0
      ABSAR=0.0
      EST=0.0
      DA=B-A
      CALL FUN(A,YA)
      C0=(A+B)/2.
      CALL FUN(C0,YM)
      CALL FUN(B,YB)
C     I=RECUR
10    LVL=LVL+1
      DX(LVL)=DA/3.
      SX=DX(LVL)/6.
      C1=A+DX(LVL)/2.
      CALL FUN(C1,Y1)
      X2(LVL)=A+DX(LVL)
      CALL FUN (X2(LVL),Y2(LVL))
      X3(LVL)=X2(LVL)+DX(LVL)
      CALL FUN (X3(LVL),Y3(LVL))
      EPSP(LVL)=EPS
      C4=DX(LVL)/2.+X3(LVL)
      CALL FUN(C4,Y4(LVL))
      YMP(LVL)=YM
      YBP(LVL)=YB
      EST1=SX*(YA+4.*Y1+Y2(LVL))
      EST2(LVL)=SX*(Y2(LVL)+Y3(LVL)+4.*YM)
      EST3(LVL)=SX*(Y3(LVL)+4.*Y4(LVL)+YB)
      SUM=EST1+EST2(LVL)+EST3(LVL)
      CLVL=LVL
      ESUM=(16.*CLVL+104.)/(6.*3.**CLVL)
      ABSAR=ABSAR-ABS(EST)+ABS(EST1)+ABS(EST2(LVL))+
     1ABS(EST3(LVL))
      IF(ABS(EST-SUM)-EPSP(LVL)*ABSAR)20,20,30
30    IF(LVL-30)40,20,20
C     BACKUP
20    LVL=LVL-1
      L=NR(LVL)
      EPVAL(LVL,L)=ESUM
      WRITE (61,101) ESUM,LVL,L
101   FORMAT ( 10X,' ESUM = ',E10.3,' LVL = ',I3,' L = ',I3)
      PVAL(LVL,L)=SUM
      GO TO (11,12,13)L
C     11=R1,12=R2,13=R3
40    NR(LVL)=1
      EST=EST1
      YM=Y1
      YB=Y2(LVL)
70    EPS=EPSP(LVL)/1.7
      DA=DX(LVL)
      GO TO 10
11    NR(LVL)=2
      YA=Y2(LVL)
      YM=YMP(LVL)
      YB=Y3(LVL)
```

```
      EST=EST2(LVL)
      A=X2(LVL)
      GO TO 70
12    NR(LVL)=3
      YA=Y3(LVL)
      YM=Y4(LVL)
      YB=YPP(LVL)
      EST=EST3(LVL)
      A=X3(LVL)
      GO TO 70
13    SUM=PVAL(LVL,1)+PVAL(LVL,2)+PVAL(LVL,3)
      ESUM=EPVAL(LVL,1)+EPVAL(LVL,2)+EPVAL(LVL,3)+5.
      WRITE (61,102) ESUM
 102  FORMAT ( 10X, ' TESUM= ',E9.2)
      IF (LVL-1) 50,50,20
50    CONTINUE
      END
```

L. Fortran Program for GAUSS

```
      PROGRAM GAUSS
C     GAUSS QUADRATURE
      DIMENSION R(5),U(5)
      M=2
103   A=1.0
      B=2.0
      FI=0.0
      GO TO (1,2,3,4,5,6,7,8,9,10,101) M
20    NM=M/2
      L=(M+1)/2
      FN=M-2*NM
      A1=B-A
      A2=A+B
      DO 100 I=1,NM
      X1=(A2+A1*U(I))/2.
      X2=(A2-A1*U(I))/2.
      CALL FUN(X1,Y1)
      CALL FUN(X2,Y2)
100   FI=FI+R(I)*(Y1+Y2)
      CALL FUN (A2/2.,Y3)
      SUM=A1/2.*(FI+FN*R(NM+1)*Y3)
1     GO TO 99
2     U(1)=0.57735026919
      R(1)=1.0
      GO TO 20
3     U(1)=0.77459666924
      U(2)=0.0
      R(1)=0.55555555556
      R(2)=0.88888888889
      GO TO 20
4     U(1)=0.86113631159
      U(2)=0.33998104359
      R(1)=0.34785484514
      R(2)=0.65214515486
      GO TO 20
5     U(1)=0.90617984594
      U(2)=0.53846931011
      U(3)=0.0
      R(1)=0.23692688506
      R(2)=0.47862867050
      R(3)=0.56888888889
      GO TO 20
6     U(1)=0.93246951420
      U(2)=0.66120938647
      U(3)=0.23861918608
      R(1)=0.17132449239
      R(2)=0.36076157305
      R(3)=0.46791393457
      GO TO 20
7     U(1)=0.94910791234
      U(2)=0.74153118560
      U(3)=0.40584515133
      U(4)=0.0
      R(1)=0.12948496617
      R(2)=0.27970539149
      R(3)=0.38183005051
      R(4)=0.41795918367
      GO TO 20
8     U(1)=0.96028985650
      U(2)=0.79666647741
      U(3)=0.52553240992
```

```
      U(4)=0.19343464250
      R(1)=0.10122855630
      R(2)=0.22238103445
      R(3)=0.31379664589
      R(4)=0.36268378338
      GO TO 20
9     U(1)=0.96816023951
      U(2)=0.83603110733
      U(3)=0.61337143270
      U(4)=0.32425342340
      U(5)=0.0
      R(1)=0.08127439836
      R(2)=0.18064816069
      R(3)=0.26061069640
      R(4)=0.31234707704
      R(5)=0.33023935500
      GO TO 20
10    U(1)=0.97390652852
      U(2)=0.86526336669
      U(3)=0.67940956830
      U(4)=0.43339539413
      U(5)=0.14887433898
      R(1)=0.06667134431
      R(2)=0.14945134915
      R(3)=0.21908636252
      R(4)=0.26926671931
      R(5)=0.29552422471
      GO TO 20
99    CONTINUE
      ERR=SUM-0.50
      IF (ABS(ERR)-0.0000000001) 101,101,102
102   M=M+1
      GO TO 103
101   CONTINUE
      WRITE (61,11) SUM,M
11    FORMAT ( ' AREA = ',E20.10,' M = ',I3)
      END
      SUBROUTINE FUN(X,Y)
      Y=1./(X*X)
      END
```

M. Fortran Program for RTGAUSS5

```
      PROGRAM REGAUSS5
C     REPEATED GAUSS QUADRATURE 5
      DIMENSION R(3),U(3)
      A=1.0
      B=2.0
      M=5
      R(1)=0.23692688506
      R(2)=0.47862867050
      R(3)=0.56888888889
      U(1)=0.90617984594
      U(2)=0.53846931010
      N=1
      DX=B-A
      A0=A
30    B=A0
      SUM=0.0
      DO 20 J=1,N
      FI=0.0
      A=B
      B=A+DX
      A1=DX
      A2=A+B
      DO 100 I=1,2
      X1=(A2+A1*U(I))/2.
      X2=(A2-A1*U(I))/2.
      CALL FUN (X1,Y1)
      CALL FUN (X2,Y2)
100   FI=FI+R(I)*(Y1+Y2)
      CALL FUN (A2/2.,Y3)
      SUM1=A1/2.*(FI+R(3)*Y3)
20    SUM=SUM+SUM1
      ERR=SUM-3.50
      IF (ABS(ERR)-0.00000001) 110,110,120
120   DX=DX/2.
      N=2*N
      GO TO 30
110   CONTINUE
      CALL PRINT2(N,SUM)
      END
```