NAG-250 10

AN ANNUAL REPORT

A POLYMORPHIC RECONFIGURABLE EMULATOR FOR PARALLEL SIMULATION

Submitted to:

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Submitted by:

E. A. Parrish, Jr.
Professor and Chairman

E. S. McVey
Professor

G. Cook
Professor

Department of Electrical Engineering
RESEARCH LABORATORIES FOR THE ENGINEERING SCIENCES
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA

Report No. UVA/528173/EE80/102
April 1980

FOREWORD

This report summarizes research performed on the PREPS project during the first year. The project has supported three half-time graduate students (2 Ph. D. and 1 M.E.) and three principal investigators (Professor G. Cook was on leave from 1 September 1979 through 3 May 1980) at 10% effort.

SECTION 1

INTRODUCTION

This research is directed at trying to capitalize on the recent developments in microprocessors and arithmetic support chips to design a reconfigurable emulator for real time flight simulation. The system consists of (1) a Master Control System (MCS) to perform all man/machine interactions and to configure the hardware to emulate a given aircraft, and (2) numerous Slave Compute Modules (SCMs) which comprise the parallel computational units.

Three specific problem areas are reported on herein. First, early work has shown that all parts of the state equations can be worked on simultaneously but that the algebraic equations cannot (unless they are slowly varying). Thus, Section 2 reports on attempts to obtain new algorithms that will allow parallel updates.

The second area involves determining the word length and step size to be used in the SCMs. This work is described in Section 3 as well as [1].

Finally, the architecture of the hardware and software is covered in Section 4 and in [2]. The architecture must clearly reflect the outcome of the above studies as well as conform to the conceptual framework of PREPS.

SECTION 2

ALGORITHMS FOR PARALLEL PROCESSING

2.1 Introduction

It is a general rule that "A more accurate method is more time-consuming," but in a real-time simulation time is very critical. The advent of cheap and fast microprocessors suggests that we can solve problems by using multiprocessors in parallel.

Now the question is, can we use multiprocessor systems to solve numerical differential equations such as

$$\dot{x} = f(x,t) \tag{2-1}$$

Looking at some well-known numerical integration methods, such as the Euler method

$$x_{n+1} = x_n + Tf(x_n), \tag{2-2}$$

or the AB-2 method

$$x_{n+1} = x_n + \frac{T}{2}[3f(x_n) - f(x_{n-1})], \tag{2-3}$$

we see that we cannot obtain x_{n+1} before obtaining the value of $f(x_n)$, the two tasks have to be done in series. Unfortunately, to update $f(x_n)$ is time-consuming even for moderate complexity of $f(x)$. We need new algorithms that can update x and $f(x)$ at the same time. If we consider only numerical integration algorithms in the form

$$x_{n+1} = x_n + T[f(x_{n-1}), f(x_{n-2}), \ldots.] \tag{2-4}$$

we can calculate $f(x_n)$ and x_{n+1} almost simultaneously. That is, they can be done in parallel. In the following section we develop several algorithms. We also have determined their corresponding stability regions

and tried to enlarge them. The idea appears to be very promising as a means of exploiting parallel processing in digital simulation.

2.2 Development

The general form of k step multistep integration algorithms is

$$x_{n+1} = \sum_{i=k}^{n} [a_i x_i + T\beta_i f_i] \qquad (2\text{-}5)$$

where n, k are integers $k < n$. After setting $a_n = 1$, $\beta_n = 0$ and $a_i = 0$ for $k \leq i < n$, we can develop several new algorithms by the coefficient-matching method for different values of k. For stability considerations we only take k = 2,3.

Assuming k = 2, we have the basic form

$$x_{n+1} = x_n + T[\beta_1 f_{n-1} + \beta_2 f_{n-2}], \qquad (2\text{-}6)$$

or

$$x_{n+3} = x_{n+2} + T[\beta_1 \dot{x}_{n+1} + \beta_2 \dot{x}_n], \text{ since } \dot{x} = f(x,t). \qquad (2\text{-}6)$$

Step 1: Using the Taylor Series Expansion, we have

$$x_{n+3} = x_n + 3Tx'_n + \frac{(3T)}{2} x''_n + \ldots. \qquad (2\text{-}7)$$

$$x_{n+2} = x_n + 2Tx'_n + \frac{(2T)}{2!}x''_n + \frac{(3T)}{3!}x'''_n + \ldots \qquad (2\text{-}8)$$

$$T\beta_1 x'_{n+1} = T\beta_1 x'_n + T^2\beta_1 x''_n + T^3\beta_1 \frac{1}{2!} x'''_n + \ldots. \qquad (2\text{-}9)$$

$$T\beta_2 \dot{x}_n = T\beta_2 \dot{x}_n \qquad (2\text{-}10)$$

Step 2: Use coefficient-matching method to minimize the local truncation error.

The minimum local truncation error occurs for $\beta_1 = \frac{5}{2}$ and $\beta_2 = -\frac{3}{2}$ and is

$$E_{LT} = \frac{23}{12} T^3 x_n''' + 0(T^4), \tag{2-11}$$

and

$$x_{n+1} = x_n + \frac{T}{2}[5f_{n-1} - 3f_{n-2}]. \tag{2-12}$$

By the same procedures, we can get another algorithm for k = 3, i.3.,

$$x_{n+1} = x_n + \frac{T}{12}[53f_{n-1} - 64f_{n-2} + 23f_{n-3}] \tag{2-13}$$

and

$$E_{LT} = \frac{55}{24} T^4 x_n^2 + 0\ (T^5) \tag{2-14}$$

These algorithms are listed together with two other non-minimum local truncation error algorithms in Table 2-1.

Now these algorithms can be implemented in parallel, because when we update x_{n+1}, we only need to utilize f_{n-1} for which the update was begun just after getting x_{n-1}. Although this cannot be done completely in parallel, it gives us a hope of the further development.

2.3 Stability

Stability is another improtant criterion in evaluating the usefulness of a numerical integration algorithm. Any acceptable algorithm must be at least weakly stable, usually we need strong stability. The stability regions of the above algorithms, compared with AB methods, are shown in Figure 2.1.

Table 2-1. Truncation Errors of Several Algorithms

Name	Algorithm	Local Truncation Error
P1	$x_{n+1} = x_n + \frac{T}{2}[5f_{n-1} - 3f_{n-2}]$	$E_{LT} = \frac{23}{12} T^3 x_n''' + O(T^4)$
P2	$x_{n+1} = x_n + \frac{T}{2}[8f_{n-1} - 9f_{n-2} + 3f_{n-3}]$	$E_{LT} = \frac{5}{12} T^3 x_n''' + O(T^4)$
P3	$x_{n+1} = x_n + \frac{T}{24}[51f_{n-1} - 18f_{n-2} - 9f_{n-3}]$	$E_{LT} = \frac{55}{24} T^3 x_n'''' + O(T^4)$
P4	$x_{n+1} = x_n + \frac{T}{24}[53f_{n-1} - 64f_{n-2} + 23f_{n-3}]$	$E_{LT} = \frac{55}{24} T^4 x_n^{IV} + O(T^5)$

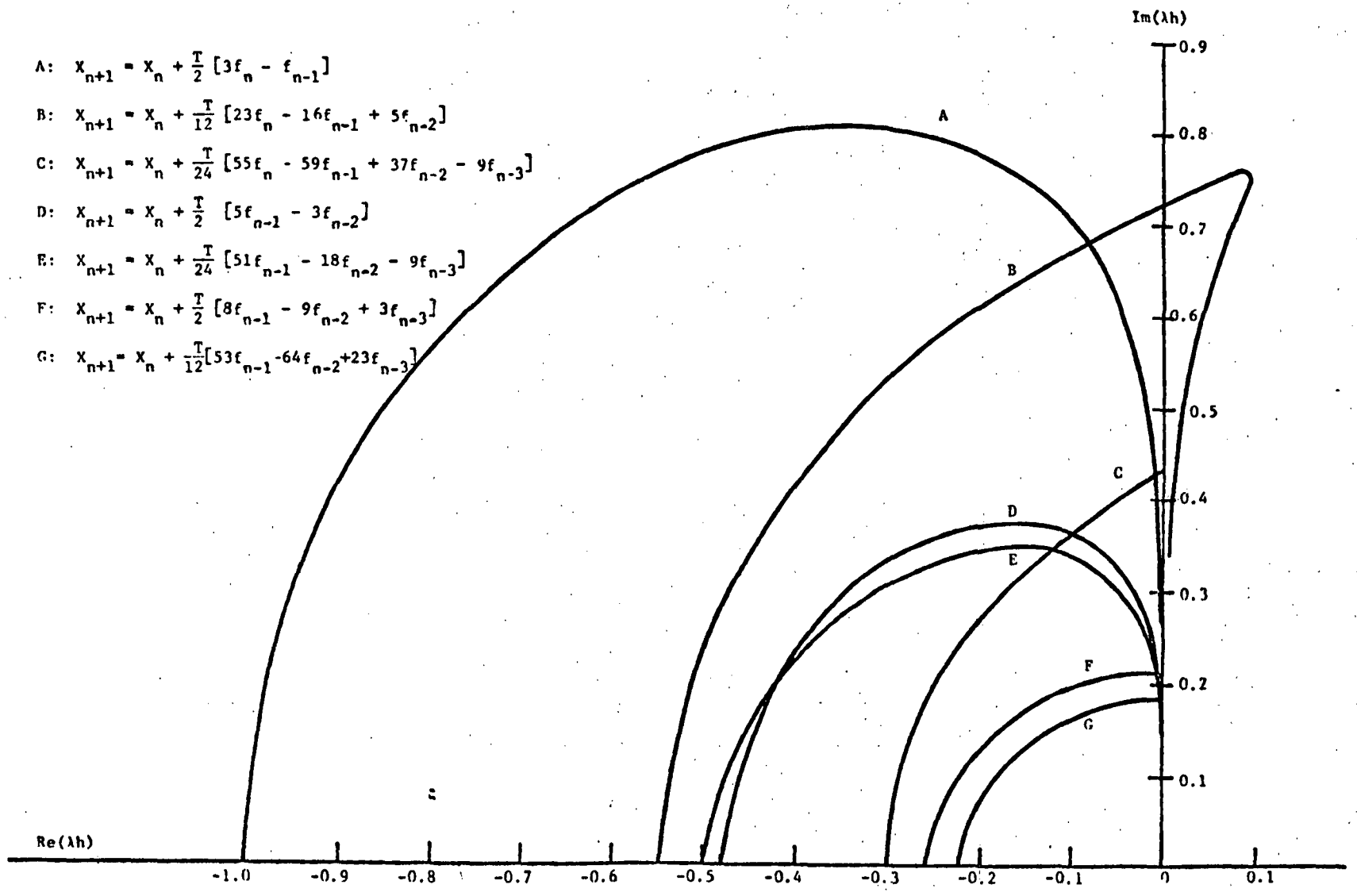

Figure 2-1. Stability regions for P1, P2, P3, P4 and AB methods

Based on accuracy considerations, P4 is the algorithm that we are interested in, but it has a quite small stability region. It means that if we want to use it, we must pick a step size sufficiently small to force the value of λh to fall into the stability region and this really limits its usefulness.

A new question arises: "If we change the basic form into

$$x_{n+1} = Ax_n + Bx_{n-1} + Cx_{n-2} + Dx_{n-3} + T[Ef_{n-1} + Ff_{n-2} + Gf_{n-3}] \quad (2\text{-}15)$$

could we change the situation?" After further investigation, we found we could enlarge the stability region a bit, but not much (Detail see Appendix in Section 2.5). This algorithm is therefore confined to solving differential equations with small λ if a large h is to be used.

2.4 Summary

In the previous section we have developed several new numerical integration algorithms. These new algorithms make it possible for us to update x_{n+1} and f_{n+1} simultaneously. Thus the algorithm can be implemented via a set of fast and low cost parallel processors. The stability regions have been studied and plotted. In all cases the range of stability was reduced by the use of stale data. Nevertheless the stable range was still usually large enough to accomodate a value for T of one-fifth the time constant ($T/2 = .2$ or $\lambda T = .2$) which is as large as one normally wishes to use.

2.5 Appendix

THEOREM:

A necessary condition for stability of the linear multistep method

$$\alpha_k y_{n+k} + \alpha_{k-1} y_{n+k-1} + \ldots + \alpha_o y_n = T\{\beta_k f_{n+k} + \beta_{k-1} f_{n+k-1} + \ldots + \beta_o f_n\} \tag{2-16}$$

is that no root of the associated polynomial

$$P(\delta) = \alpha_k \delta^k + \alpha_{k-1} \delta^{k-1} + \ldots + \alpha_o \tag{2-17}$$

may have modules which exceed 1, and that the roots of modulus 1 be simple.

Since the general form is

$$x_{n+1} = Ax_n + Bx_{n-1} + Cx_{n-2} + Dx_{n-3} + T[Ef_{n-1} + Ff_{n-2} + Gf_{n-3}] \tag{2-18}$$

where A, B, C, D, E, F, G are coefficients to be determined and T is step size, the first constraint, due to the theorem, is that the polynomial

$$P(\delta) = \delta^4 - A\delta^3 - C\delta - D \tag{2-19}$$

has no roots with modulus exceeding 1 and that those of modulus 1 be simple.

Using the maximum coefficient-matching method, we can solve for E, F, G in terms of A, B, C, D as follows:

$$E = \frac{1}{12}[81 - 28A - 5B - D] \tag{2-20}$$

$$F = \frac{1}{3}[2A - 2B + 2D - 18] \tag{2-21}$$

$$G = \frac{9}{4} - \frac{1}{3} A + \frac{1}{12} B + \frac{5}{12} D \tag{2-22}$$

and local truncation error also depends on A, B, C, D,

$$E_{LT} = (\frac{8}{3} - \frac{3}{8} A - \frac{1}{24} C)T^4 x_n + 0(T^5) \qquad (2\text{-}23)$$

Now all we have to do is to decide the possible values for A, B, C, D to meet the constraint. We put one root at 0, another at 1, just as all the other numerical methods do, and let the other two roots δ_1, δ_2 move around within a unit circle. The problem can be divided into two cases:

Case 1: δ_1, δ_2 are on the real axis, i.e., $\delta_1 = k_1$, $\delta_2 = k_2$ and

$$|k_1| < 1, \quad |k_2| < 1$$

Equation (2-19) becomes

$$\delta^4 - (k_1+k_2+1)\delta^3 + (k_1k_2+k_1+k_2)\delta^2 + k_1k_2\delta = 0, \qquad (2\text{-}24)$$

and

$$A = (k_1+k_2+1) \qquad (2\text{-}25)$$

$$B = -(k_1k_2+k_1+k_2) \qquad (2\text{-}26)$$

$$C = k_1k_2 \qquad (2\text{-}27)$$

$$D = 0 \qquad (2\text{-}28)$$

The local truncation error is

$$E_{LT} = \frac{55}{24} - \frac{3}{8} (k_1+k_2) - \frac{1}{24}(k_1k_2). \qquad (2\text{-}29)$$

Case 2: δ_1, δ_2 are complex conjugate, i.e., $\delta_1 = \alpha+j\omega$, $\delta_2 = \alpha-j\omega$

and

$$\sqrt{\alpha^2+\omega^2} < 1.$$

Equation (2-19) becomes

$$\delta^4 - (2\alpha+1)\delta^3 + (\alpha^2+\omega^2+2\alpha)\delta^2 + (\alpha^2 + \omega^2) = 0 \quad (2\text{-}30)$$

and

$$A = 2\alpha+1 \quad (2\text{-}31)$$

$$B = -(\alpha^2+\omega^2+2\alpha) \quad (2\text{-}32)$$

$$C = \alpha^2+\omega^2 \quad (2\text{-}33)$$

$$D = 0 \quad (2\text{-}34)$$

The local truncation error is

$$E_{LT} = \frac{55}{24} - \frac{3}{4}\alpha - \frac{1}{24}(\alpha^2+\omega^2). \quad (2\text{-}35)$$

From Figures 2-2 to 2-10, the stability regions for different values of δ1 and δ_2, we can see that for most cases the stability region is larger than that of algorithm P4 and the stability regions get bigger and bigger when δ_1 and/or δ_2 approach the boundary of the unit circle. But we know all the existing stable multistep methods have δ_1 and δ_2 at the origin, and when δ_1 and δ_2 approach the boundary of the unit circle, we would expect the algorithm to be less stable or less accurate. We need to do further investigation.

Assume $\dot{x} = -\lambda x$ where $\lambda < 0$. Then the difference equation

$$x_{n+1} - Ax_n - [\beta-\lambda TE\, x_{n-1}] - [C-\lambda TF]x_{n-2} + \lambda TGx_{n-3} = 0 \quad (2\text{-}36)$$

has a solution of the form

$$x_n = C_1\beta_1^n + C_2\beta_2^n + C_3\beta_3^n + C_4\beta_4^n. \quad (2\text{-}37)$$

As $T \to 0$, $\beta_1 = 1$, $\beta_2 = 0$, $\beta_3 = k_1$, $\beta_4 = k_2$. Here β_1 represents the true root, and β_2, β_3 and β_4 are extraneous roots which arise only because we have replaced a first order differential equation by a fourth-order difference equation. For a strong stable algorithm, we have to let $\beta_2 < 1$, $\beta_3 < 1$, $\beta_4 < 1$.

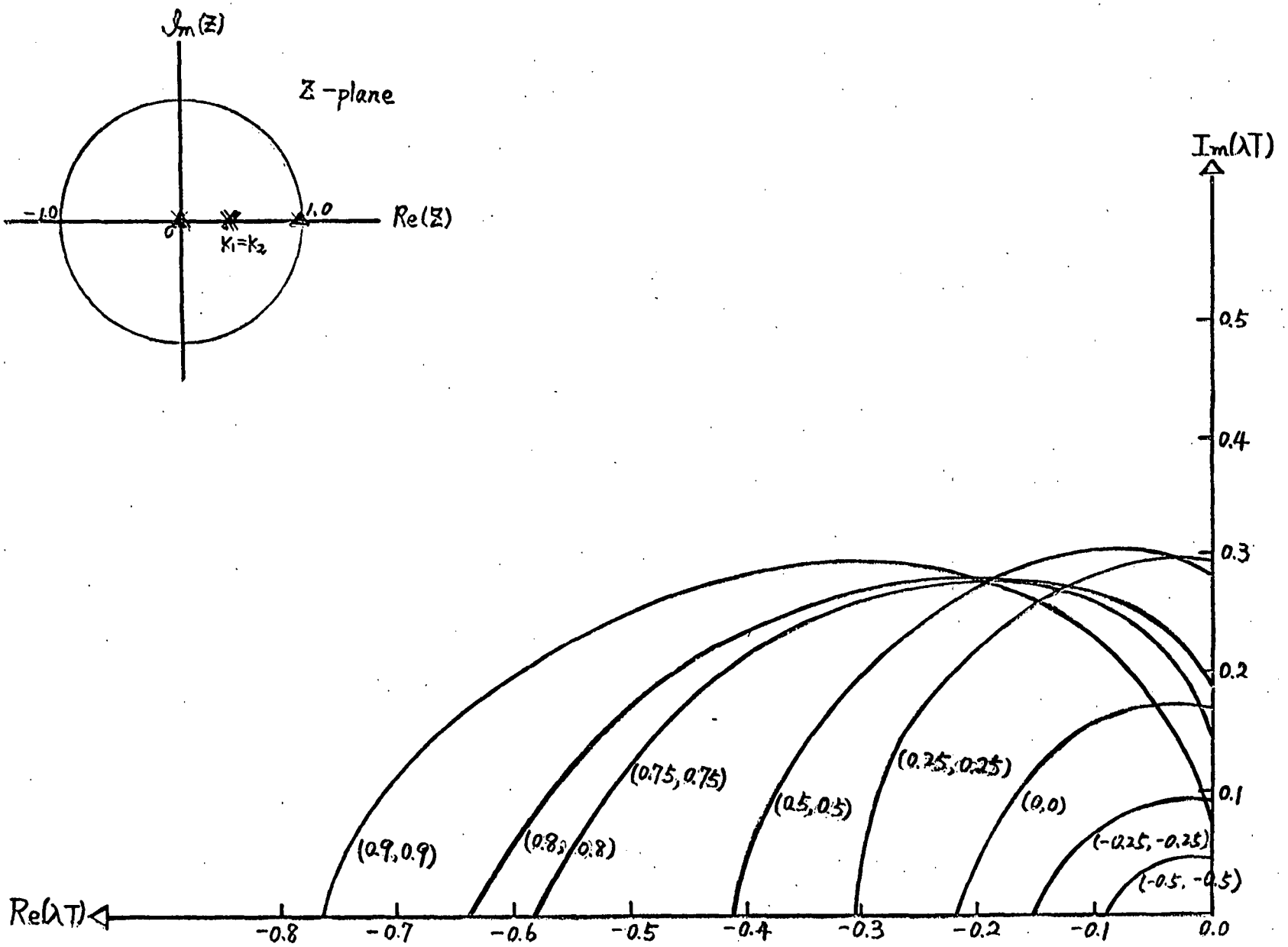

Figure 2-2. Stability regions for $k_1 = k_2$ on the Re(z) axis

*(0,0) denote the stability plot for P4

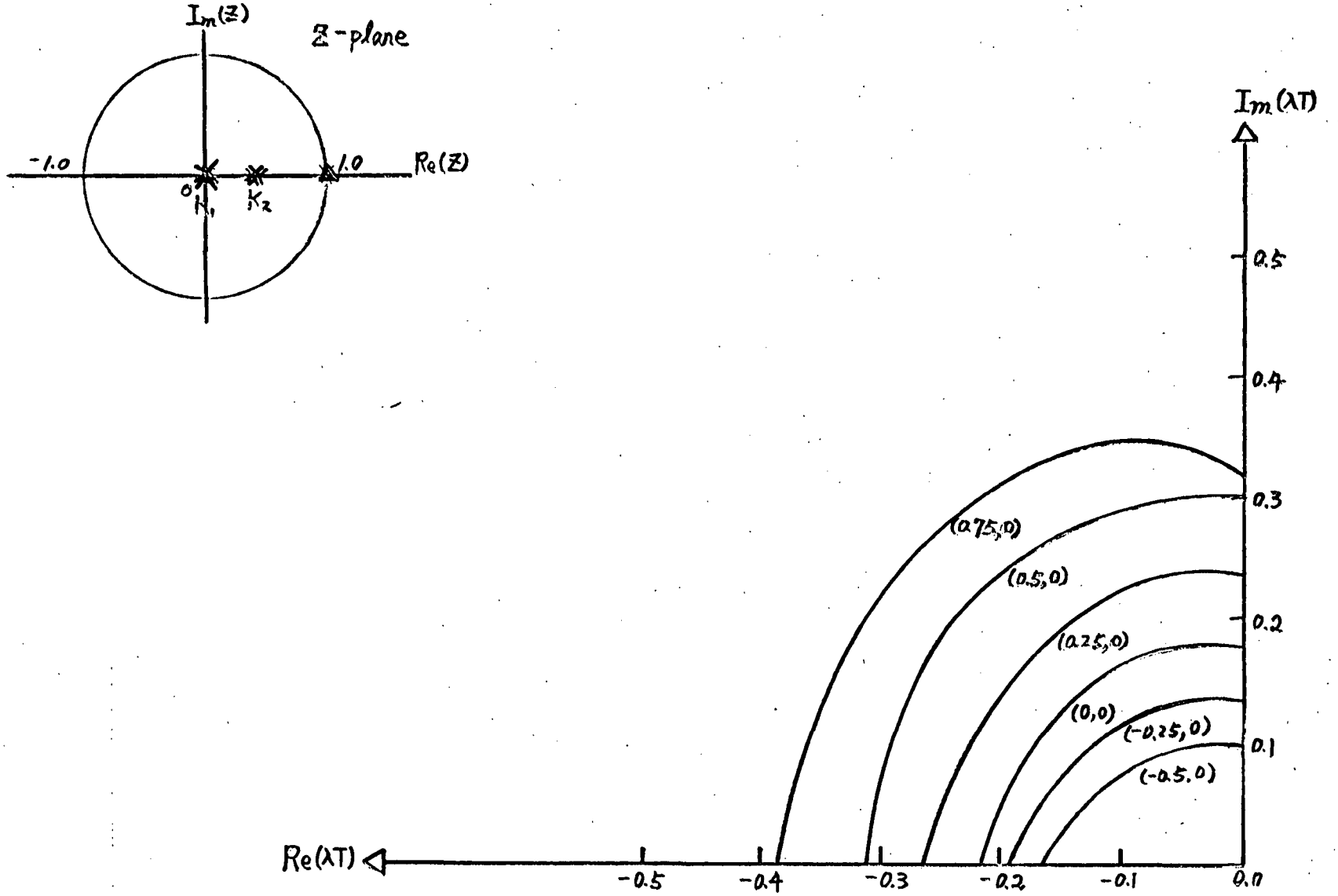

Figure 2-3. Stability regions for $k_1 = 0$, k_2 on the Re(z) axis

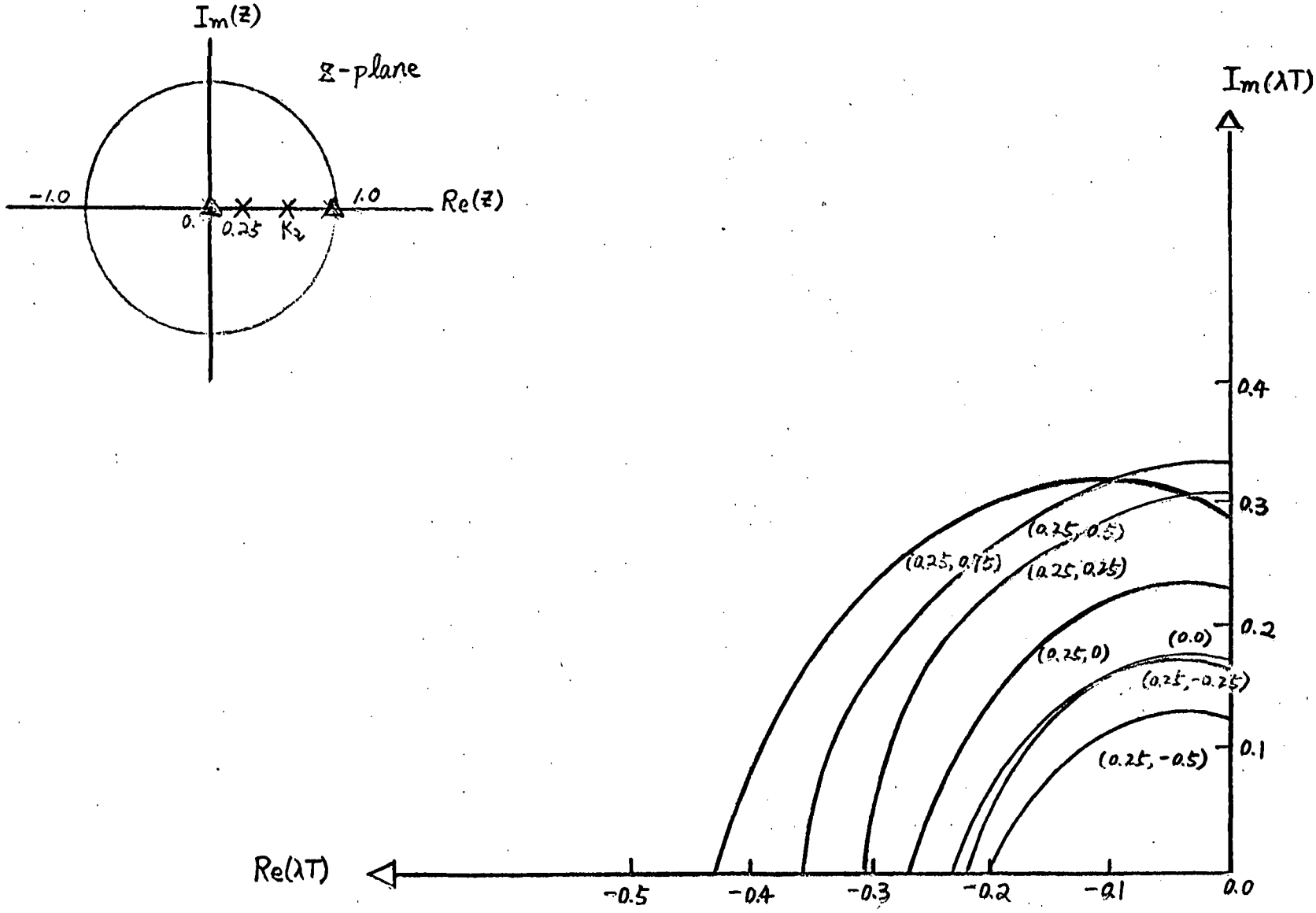

Figure 2-4. Stability regions for $k_1 = 0.25$, k_2 on the Re(z) axis

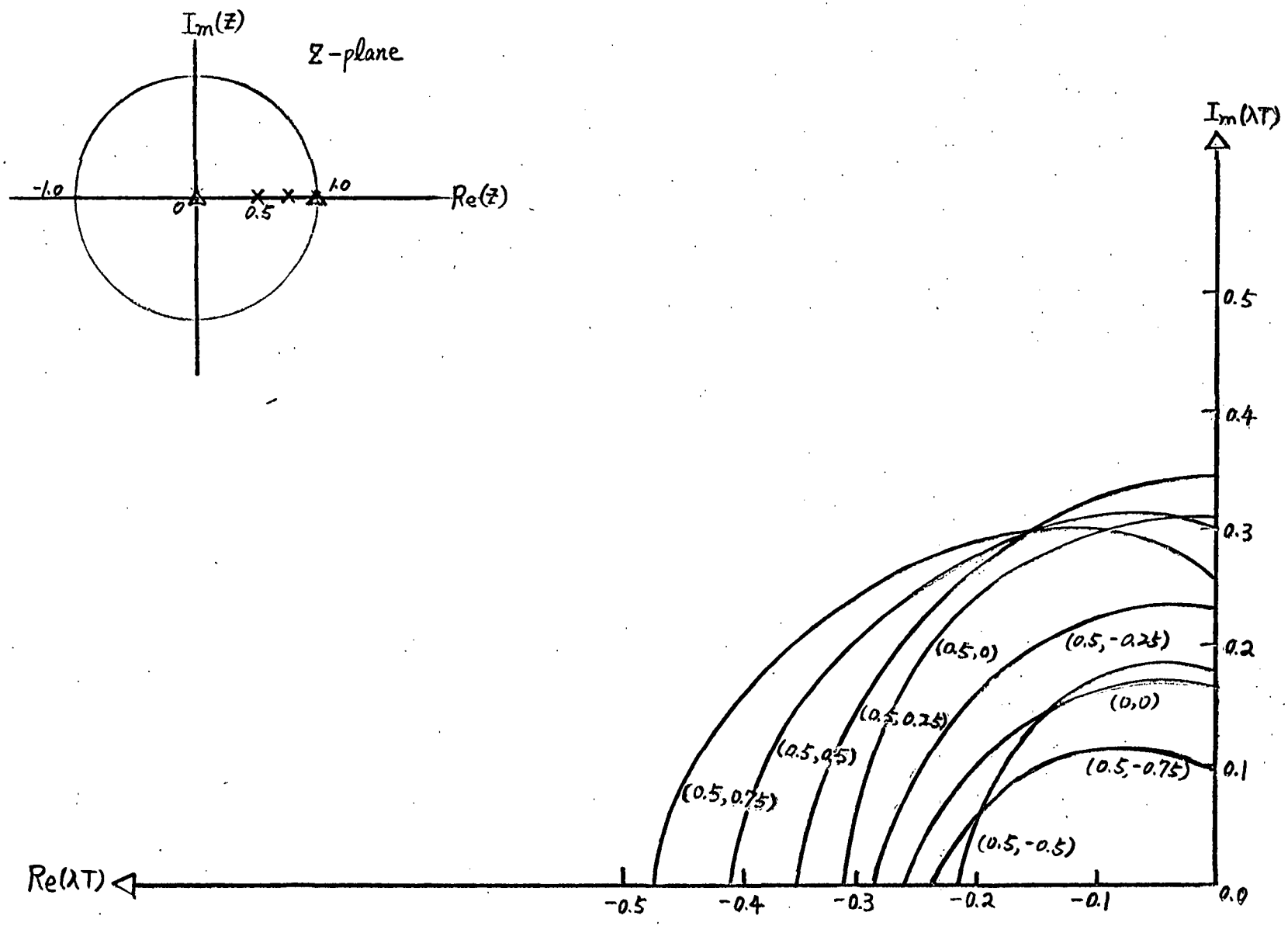

Figure 2-5. Stability regions for $k_1 = 0.5$, k_2 on the Re(z) axis

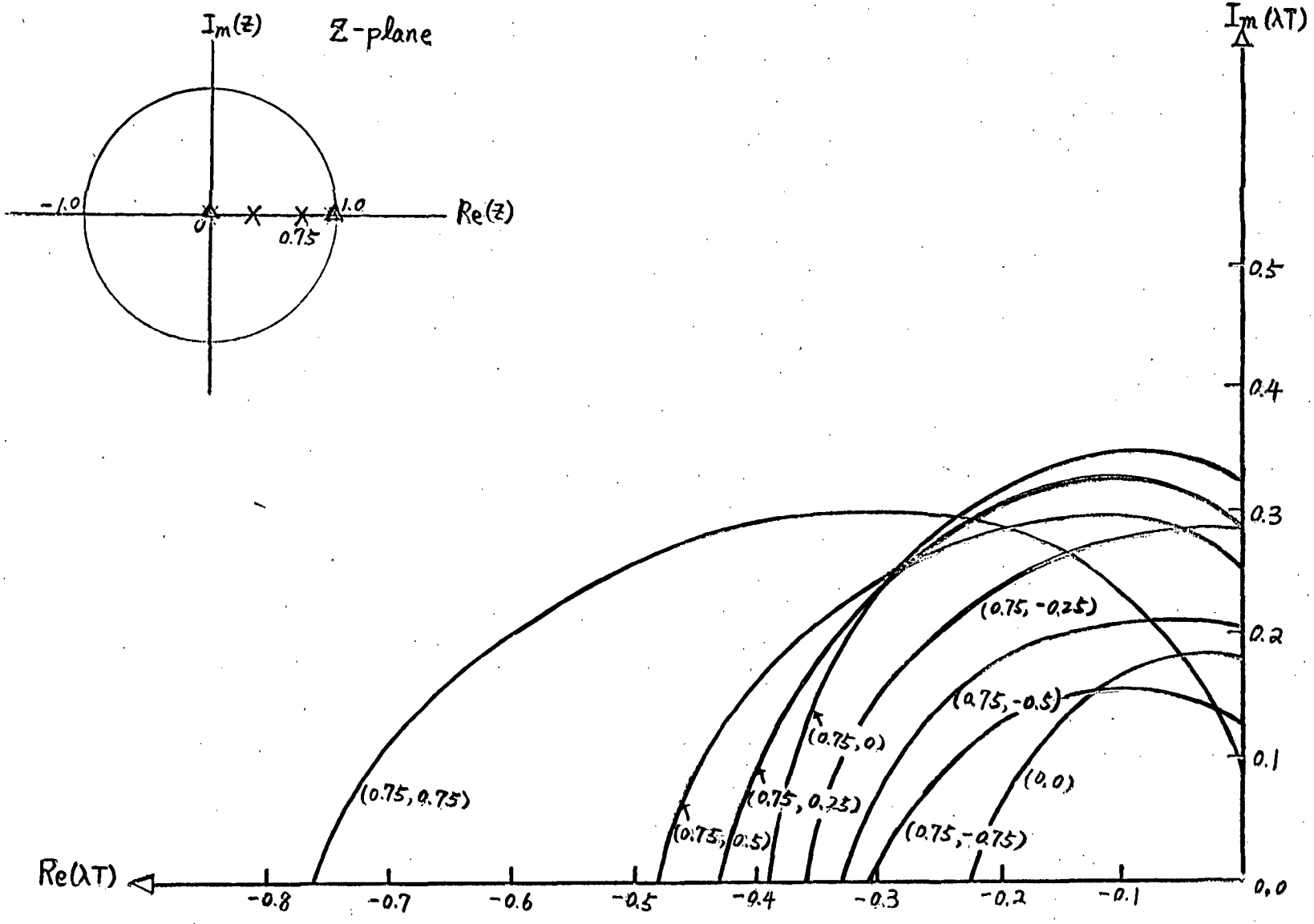

Figure 2-6. Stability regions for $k_1 = 0.75$, k_2 on the Re(z) axis

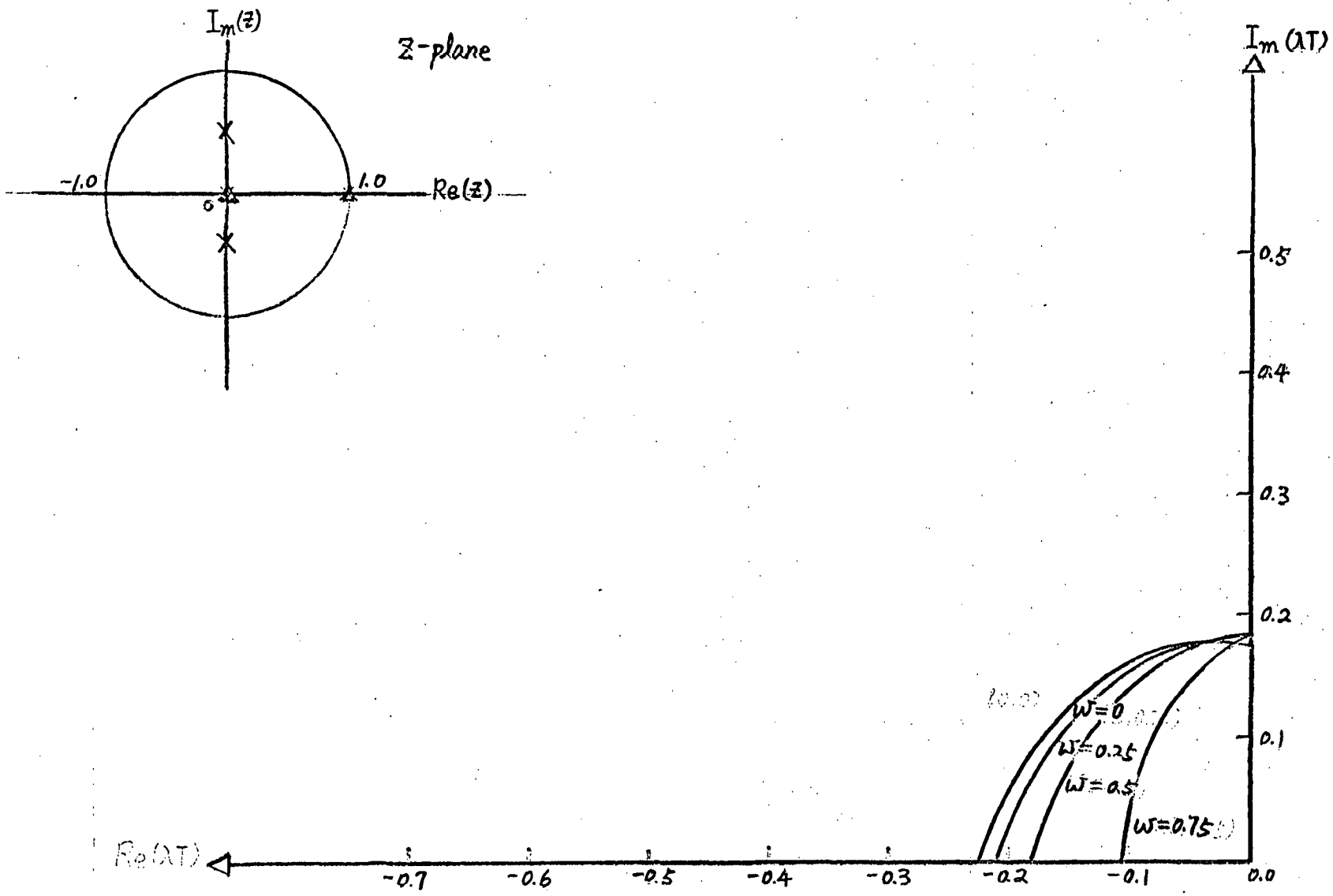

Figure 2-7. Stability regions for complex conjugate pair on Im(z) axis

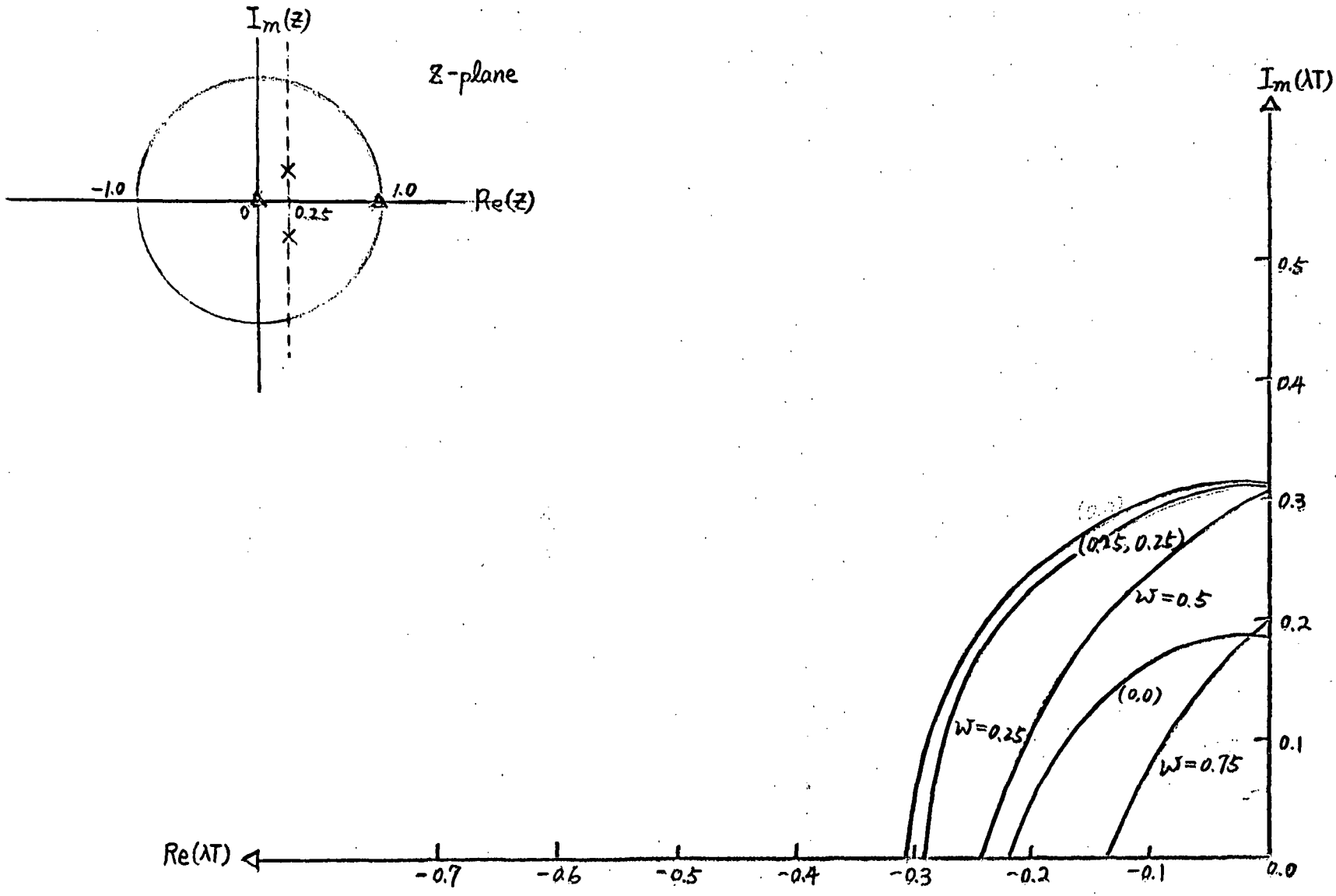

Figure 2-8. Stability region for $k_1 = 0.25 + j\omega$, $k_2 = 0.25 - j\omega$

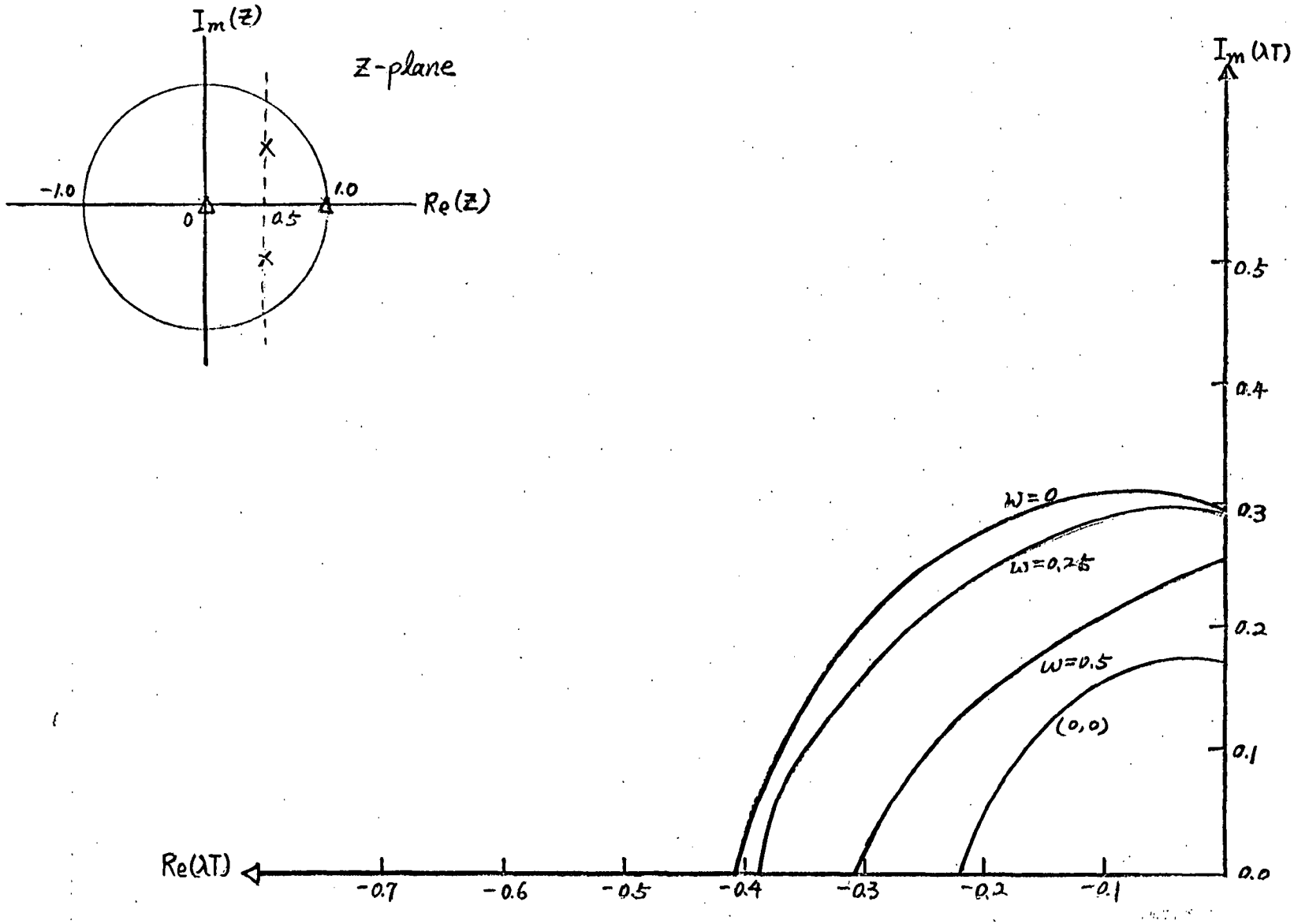

Figure 2-9. Boundary region for $k_1 = 0.5 + j\omega$, $k_2 = 0.5 - j\omega$

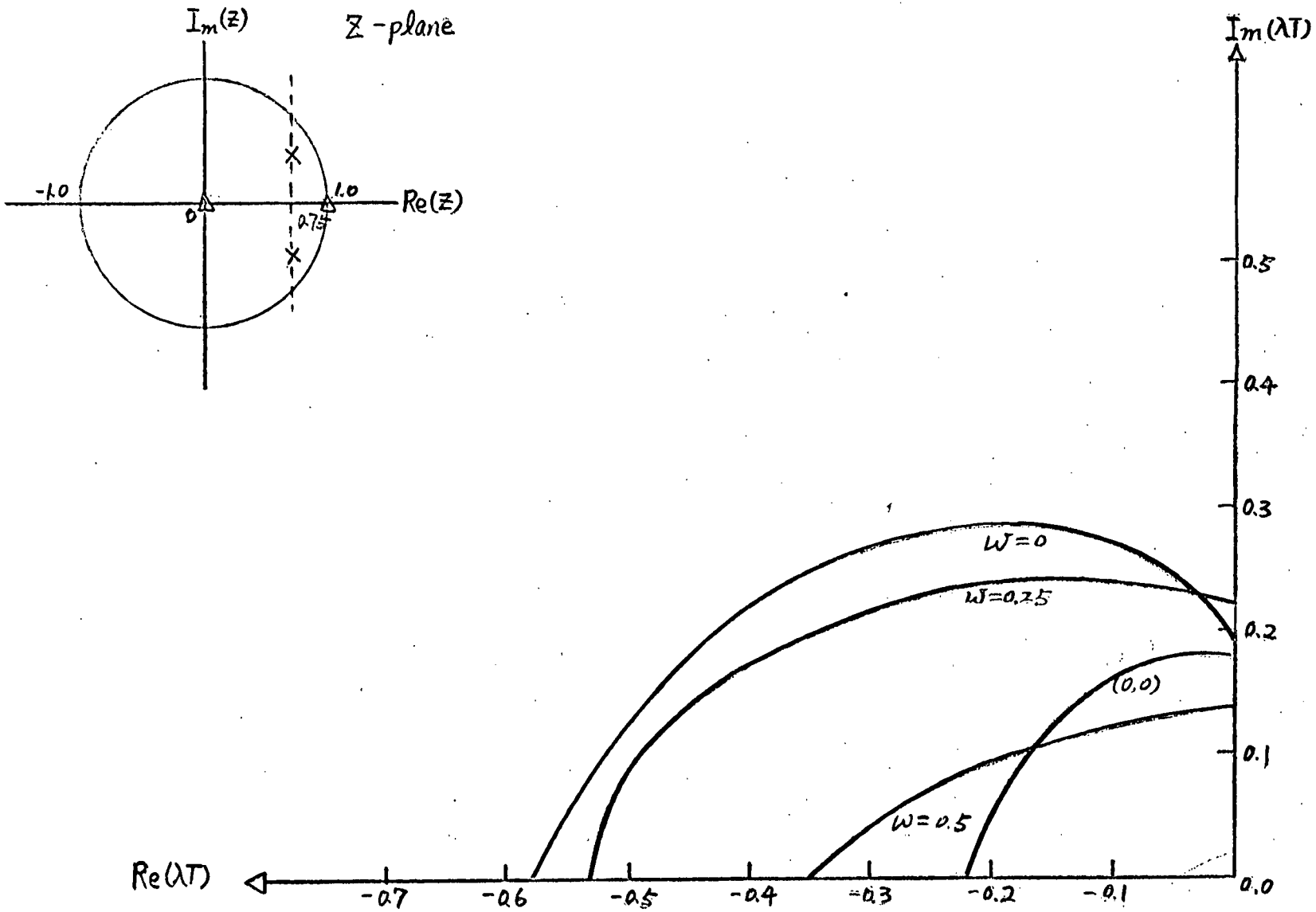

Figure 2-10. Stability region for $k_1 = 0.75 + j\omega$, $k_2 = 0.75 - j\omega$

From Tables 2-2 to 2-7, which are constructed by varying the value of λT, = -0.27 is the upper bound for stability and accuracy consideration, because at this value some algorithms have one root greater than unity and others lose accuracy by introducing imaginary parts. If we want a little greater stability to suit a large λT, there are two choices:

(i) $$x_{n+1} = \frac{5}{4} x_n - \frac{1}{4} x_{n-1} + \frac{T}{48}[189f_{n-1} - 256f_{n-2} + 87f_{n-3}] \quad (2\text{-}38)$$

or

$$E_{LT} = \frac{211}{96} T^4 x_n^{IV} + 0(T^5) \quad (2\text{-}39)$$

(ii) $$x_{n+1} = 2x_n - \frac{3}{2} x_{n-1} + \frac{1}{2} x_{n-2} + \frac{T}{24}[65f_{n-1} - 88f_{n-2} + 35f_{n-3}] \quad (2\text{-}40)$$

$$E_{LT} = \frac{91}{48} T^4 x_n^{IV} + 0(T^5) \quad (2\text{-}41)$$

We need to perform simulations to further judge their usefulness.

Table 2-2a. Root Locations

$\lambda T = -0.05$					
k_1	k_2	β_1	β_2	β_3	β_4
0	0	0.951215	0.355454	-0.15335 + j0.50983	
0.25	-0.25	0.951214	0.401046	-0.17613 + j0.469974	
0.25	0	0.95121	0.432307	-0.06676 + j0.46468	
0.25	0.25	0.951204	0.48603	-0.03138 + j0.42803	
0.25	0.5	0.951199	0.587492	0.10565 + j0.36254	
0.25	0.75	0.951124	0.765601	0.14164 + j0.28612	$\overline{\beta_3}$
0.5	-0.5	0.951209	0.542992	-0.2471 + j0.35567	
0.5	-0.25	0.951206	0.550895	-0.12605 + j0.39755	
0.5	0	0.951199	0.563707	-0.00745 + j0.39905	
0.5	0.5	0.951169	0.641303	0.203764+ j0.28235	
0.5	0.75	0.951152	0.775251	0.26180 + j0.15489	
0.75	-0.5	0.951185	0.757975	-0.22958 + j0.27415	
0.75	-0.25	0.95118	0.759343	-0.10526 + j0.32903	
0.75	0	0.951171	0.76532	-0.01865 + j0.33224	
0.75	0.75	0.951036	0.816661	0.59838 + j0.133921	

Note: $\overline{\beta_3}$ means complex conjugate of β_3.

x means the absolute value of $\beta_i > 1$.

Table 2-2b. Root Locations

$\lambda T = -0.05$					
k	ω	β_1	β_2	β_3	β_4
0	0	0.95121	0.35545	-0.15334 + j0.50983	
0	0.25	0.95122	0.310305	-0.13213 + j0.55091	
0.25	0.25	0.95121	0.40513	-0.07183 + j0.46384	$\overline{\beta_3}$
0.25	0.5	0.95121	0.22257	0.16311 + j0.60885	
0.5	0.25	0.95119	0.42949	0.30966 + j0.29071	
0.5	0.5	0.95120	0.15311	0.44784 + j0.54781	
0.75	0.25	0.95117	0.11223	0.71831 + j0.25441	

Table 2-3a. Root Locations

$\lambda T = -0.1$					
k_1	k_2	β_1	β_2	β_3	β_4
0	0	0.904584	0.426622	0.16560 + j0.68500	
0.25	-0.25	0.904565	0.462158	-0.18336 + j0.65276	
0.25	0	0.904503	0.494706	-0.07460 + j0.63206	
0.25	0.25	0.904394	0.545743	0.02493 + j0.58689	
0.25	0.5	0.904164	0.632818	0.10651 + j0.51691	
0.25	0.75	0.90306	0.784402	0.15627 + j0.43027	
0.5	-0.5	0.904471	0.576176	-0.24032 + j0.56038	$\overline{\beta_3}$
0.5	-0.25	0.904409	0.58748	-0.12094 + j0.57312	
0.5	0	0.904319	0.604521	-0.00442 + j0.55899	
0.5	0.5	0.903739	0.686548	0.20486 + j0.44321	
0.5	0.75	0.901951	0.801461	0.27329 + j0.14908	
0.75	-0.5	0.903895	0.768846	-0.21137 + j0.46971	
0.75	-0.25	0.903738	0.771819	-0.08778 + j0.48934	
0.75	0	0.903464	0.776442	0.03505 + j0.47692	
0.75	0.75	0.895464	0.855484	0.37456 + j0.14908	

Table 2-3b. Root Locations

$\lambda T = -0.10$					
k	ω	β_1	β_2	β_3	β_4
0	0	0.904584	0.426622	0.16560 + j0.68500	
0	0.25	0.904604	0.393114	-0.14886 + j0.71788	
0.25	0.25	0.90445	0.487811	0.05387 + j0.61801	
0.25	0.5	0.904566	0.340651	0.12739 + j0.72782	$\overline{\beta_3}$
0.5	0.25	0.904074	0.571984	0.26197 + j0.46520	
0.5	0.5	0.904434	0.297086	0.39924 + j0.61916	
0.75	0.25	0.902965	0.274497	0.66127 + j0.25067	

Table 2-4a. Root Locations

$\lambda T = -0.15$					
k_1	k_2	β_1	β_2	β_3	β_4
0	0	0.359341	0.476703	-0.16802 + j0.82072	
0.25	-0.25	0.859208	0.507451	-0.18333 + j0.79220	
0.25	0	0.858838	0.540661	-0.07475 + j0.76153	
0.25	0.25	0.858135	0.590594	0.025635+ j0.70952	
0.25	0.5	0.856351	0.671962	0.11084 + j0.63495	
0.25	0.75	0.831331 + j0.0105599	$\overline{\beta_1}$	0.168669+ j0.541913	
0.5	-0.5	0.858591	0.606184	-0.232687+ j0.709719	
0.5	-0.25	0.858204	0.620489	0.11435 + j0.70740	$\overline{\beta_3}$
0.5	0	0.857574	0.640453	0.00099 + j0.68305	
0.5	0.5	0.85037	0.72805	0.20944 + j0.55980	
0.5	0.75	0.841861 + j0.0308764	$\overline{\beta_1}$	0.28314 + j0.456348	
0.75	-0.5	0.852558	0.788329	-0.19544 + j0.61098	
0.75	-0.25	0.850571	0.79468	-0.07263 + j0.61425	
0.75	0	0.846422	0.805377	0.04910 + j0.59209	
0.75	0.75	0.867969 + j0.045016	$\overline{\beta_1}$	0.38203 + j0.31819	

Table 2-4b. Root Locations

$\lambda T = -0.15$					
k	ω	β_1	β_2	β_3	β_4
0	0	0.859341	0.476703	-0.16802 + j0.82072	
0	0.25	0.859451	0.447507	-0.15348 + j0.84966	
0.25	0.25	0.858543	0.542269	0.04959 + j0.73797	
0.25	0.5	0.859228	0.415615	0.11258 + j0.83304	$\overline{\beta_3}$
0.5	0.25	0.855972	0.637901	0.25306 + j0.58384	
0.5	0.5	0.858475	0.405967	0.36778 + j0.70171	
0.75	0.25	0.847451	0.614993	0.51878 + j0.29603	

Table 2-5a. Root Locations

$\lambda T = -0.20$					
k_1	k_2	β_1	β_2	β_3	β_4
0	0	0.813932	0.520066	-0.11700 + j0.93686	
0.25	-0.25	0.813359	0.548211	-0.18079 + j0.91069	
0.25	0	0.811803	0.582872	-0.07234 + j0.87228	
0.25	0.25	0.808502	0.634411	0.02854 + j0.81442	
0.25	0.5	-.795513	0.722697	0.11590 + j0.73553	
0.5	-0.5	0.810375	0.640033	-0.22520 + j0.83450	$\overline{\beta_3}$
0.5	-0.25	0.808471	0.656843	-0.10766 + j0.82156	
0.5	0	0.804923	0.681446	0.00682 + j0.78921	
0.5	0.5	0.785482 + j0.040399	$\overline{\beta_1}$	0.21452 + j0.657418	
0.75	-0.5	0.806713 + j0.0390617	$\overline{\beta_1}$	-0.18171 + j0.729575	
0.75	-0.25	0.809695 + j0.0424832	$\overline{\beta_1}$	-0.05969 + j0.72174	
0.75	0	0.814069 + j0.0466179	$\overline{\beta_1}$	0.06093 + j0.69197	

Table 2-5b. Root Locations

$\lambda T = -0.2$					
k	ω	β_1	β_2	β_3	β_4
0	0	0.813932	0.520066	-0.16670 + j0.93686	
0	0.25	0.814379	0.493282	-0.15383 + j0.96332	
0.25	0.25	0.810591	0.590268	0.94857 + j0.84099	
0.25	0.5	0.813614	0.474643	0.10587 + j0.92739	$\overline{\beta_3}$
0.5	0.25	0.794696	0.703101	0.25110 + j0.68157	
0.5	0.5	0.810778	0.489458	0.34988 + j0.782657	
0.75	0.25	0.777819 + j0.068809	$\overline{\beta_1}$	0.472181+ j0.428535	

Table 2-6a. Root Locations

$\lambda T = -0.25$					
k_1	k_2	β_1	β_2	β_3	β_4
0	0	0.764619	0.564556	x	x
0.25	-0.25	0.762269	0.592382	x	x
0.25	0	0.755181	0.632886	-0.06903 + j0.97124	
0.25	0.25	0.717892 + j0.0155761	$\overline{\beta_1}$	0.03211 + j0.90816	
0.25	0.5	0.7542 + j0.0637311	$\overline{\beta_1}$	0.1208 + j0.82517	$\overline{\beta_3}$
0.5	-0.5	0.740026	0.69649	-0.218258 + j0.944256	
0.5	-0.25	0.726368 + j0.0279864	$\overline{\beta_1}$	-0.10137 + j0.92315	
0.5	0	0.737616 + j0.047944	$\overline{\beta_1}$	0.01238 + j0.88403	

Table 2-6b. Root Locations

$\lambda T = -0.25$					
k	ω	β_1	β_2	β_3	β_4
0	0	0.764619	0.564556	x	x
0	0.25	0.766375	0.538369	x	x
0.25	0.25	0.748818	0.64882	0.05118 + j0.93331	$\overline{\beta_1}$
0.25	0.5	0.76406	0.530168	x	x
0.5	0.25	0.748561 + j0.068962	$\overline{\beta_1}$	0.25144 + j0.76737	
0.5	0.5	0.753959	0.567169	0.33944 + j0.85870	$\overline{\beta_3}$

Table 2-7a. Root Locations

$\lambda T = -0.27$					
k_1	k_2	β_1	β_2	β_3	β_4
0.25	0	0.716738	0.66852	x	x
0.27	0	0.701398	0.690966	x	x
0.28	0	0.69691 + j0.00906		-0.05691 + j0.99862	
0.3	0	0.69988 + j0.0227805	$\overline{\beta_1}$	-0.0498798 + j0.99199	$\overline{\beta_3}$

Table 2-7b. Root Locations

$\lambda T = -0.27$					
k	ω	β_1	β_2	β_3	β_4
0.25	0.25	0.697969 + j0.0265897	$\overline{\beta_1}$	0.05203 + j0.96797	
0.5	0.5	0.721195	0.605773	0.33652 + j0.8877	$\overline{\beta_3}$
0.5	0.4	0.70184 + j0.056508		0.29816 + j0.84226	

SECTION 3
COMPUTER WORD SIZE REQUIRED FOR SPECIFIED ERROR IN NUMERICAL INTEGRATION

3.1 Introduction

This section deals with computational errors; a question that has to be examined in the light of the particular characteristics of microcomputers, mainly their short word length. Specifically, the accumulation of round off errors in numerical integration performed by small word computers will be investigated. This choice of effort was based on what seems to be a reasonable assumption that considerations on errors in numerical integration will be decisive in determining the minimum word length required for satisfactory simulation. Also, since one of the main (and looked for) consequences of using parallel processing on a large scale will be the possibility to lower the sampling period of all signals and in particular the step size of all integrations, the discretization (or truncation) error will no longer be a source of great concern; a fact which dramatically emphasizes the new and major role played by the accumulation of round off error in the envisaged solution. Indeed, in many cases the use of an extremely unsophisticated integration algorithm like the Euler's method would be perfectly sufficient to make the discretization error smaller than the accumulated round off error.

The effectiveness of a partial use of double precision arithmetic will be shown in this presentation and the validity of including in the arithmetic an algorithm for symmetric rounding will be discussed. Two simple criteria will be established, in a particular but typical one-dimensional problem, that relate the accumulation of round off errors and the word length of the computer. However, no final, definitive, recommendation will be given regarding the minimum word length to be used for a given maximum error in the simulation of a complex problem (that is a problem involving a n^{th} order differential equation, possibly including non-linearities). In such a case a mathematical approach to the question of the accumulation of round off errors becomes very quickly hopeless, and no other solution has been found than a comparative simulation (several simulations with different word length) of each particular system. Nevertheless the conclusion that has emerged at the end of this study is that in many instances short word computers will be adequate for simulation purposes and that it will be possible to obtain valid results using extremely short step sizes.

Assuming that the dynamic system to be emulated can be divided into blocs of simultaneous differential equations, the simulation problem can be summarized, using state space representation, as the search for the real time solution of

$$\frac{d\underline{x}}{dt} = \underline{x}' = \underline{f}(\underline{x}, \underline{u}) \tag{3-1}$$

where $\underline{x}$ is the state vector and $\underline{u}$ is the input vector. This vector equation can also be written as

$$\begin{aligned} x_1' &= f_1(x_1, x_2, \ldots, x_d, u_1, u_2, \ldots, u_d) \\ x_2' &= f_2(x_1, x_2, \ldots, x_d, u_1, u_2, \ldots, u_d) \\ &\cdots\cdots \\ x_d' &= f_d(x_1, x_2, \ldots, x_d, u_1, u_2, \ldots, u_d) \end{aligned} \tag{3-2}$$

In the projected multi-microcomputer emulator each one of those d equations in (2) would be integrated by its own dedicated computer. Thus, for each individual integrator the problem can be reduced to the integration of one differential equation

$$x_i' = f(x_i, u_i) \tag{3-3}$$

where the forcing term u_i now represents the combination of all terms in x_j and u_j in the i^{th} equation of (2), except for those terms involving the dependent variable x_i.

In this study, attention will be given first to analysing the numerical integration of (3) alone, where u_i is assumed exactly known, in order to focus on the errors introduced by the integration process itself. However, the preceeding assumption cannot be made when the equation is part of a system of simultaneous equations because of the dependence of the forcing term on the other state variables. In a second step consideration will be given to the system of equations as a whole.

3.2 Euler's Method

Euler's method to integrate (3), where the subscripts can now be forgotten, is given by the algorithm

$$x_{n+1} = x_n + T\, f(x_n, u_n) \qquad (3\text{-}4)$$

$$x_0 = x_o$$

where x_o is the initial condition and T is the step size of integration. However, this formulation does not accurately represent the relationship between the successive results of the numerical integration because it does not take into account the rounding of numbers in the various operations needed to calculate x_{n+1}. Assuming that the computer uses floating point arithmetic, a more accurate representation of that relationship is

$$\tilde{x}_{n+1} = \left\{\tilde{x}_n + [T\, \tilde{f}(\tilde{x}_n, \tilde{u}_n)]^*\right\}^* \qquad (3\text{-}5)$$

$$\tilde{x}_0 = x_o^*$$

where the symbol $\tilde{a}$ is defined as the computed value of the function or the variable a and a* indicates its rounded value (to one computer word). Formulation (5) assumes that

1) no overflow occurs at any point of the computation
2) the step size of integration T is chosen such that it can be exactly expressed by a one word number.

A more manageable form of equation (5) can be obtained by explicitly expressing the error

$$\tilde{x}_{n+1} = \tilde{x}_n + T\, f(\tilde{x}_n, \tilde{u}_n) + e_{n+1} \qquad (3\text{-}6)$$

where e_{n+1} *is defined as the local round off error [1]. It can be* evaluated by

$$\begin{aligned} e_{n+1} &= \left\{\tilde{x}_n + [T\, \tilde{f}(\tilde{x}_n, \tilde{u}_n)]^*\right\}^* - [\tilde{x}_n + T\, f(\tilde{x}_n, \tilde{u}_n)] \\ &= \left\{\tilde{x}_n + [T\, \tilde{f}(\tilde{x}_n, \tilde{u}_n)]^*\right\}^* - \left\{\tilde{x}_n + [T\, \tilde{f}(\tilde{x}_n, \tilde{u}_n)]^*\right\} \\ &\quad + [T\tilde{f}(\ldots)]^* - T\tilde{f}(\ldots) + T\tilde{f}(\ldots) - Tf(\ldots) \end{aligned} \qquad (3\text{-}7)$$

tely appears that if the step size T is chosen equivalent to
tive or negative power of 2 the multiplication of the function
y T will not introduce any round off error. Then

$$[T\,\tilde{f}(\tilde{x}_n, \tilde{u}_n)]^* = T\,\tilde{f}(\tilde{x}_n, \tilde{u}_n) \tag{3-8}$$

ing this condition fulfilled, the local round off error e_{n+1} as
ssed by (7) can be divided into two terms

$$e_{n+1} = \pi_{n+1} + \rho_{n+1} \tag{3-9}$$

ere

$$\pi_{n+1} = \{\tilde{x}_n + [T\,\tilde{f}(\ldots)]^*\}^* - \{\tilde{x}_n + [T\,\tilde{f}(\ldots)]^*\}$$

$$\rho_{n+1} = T\,\tilde{f}(\ldots) - T\,f(\ldots)$$

They are respectively known [1] as the induced error which occurs when the two numbers on the right side of (4) are added, and the inherent error which occurs in the process of computing the value of the function $f(x_n, u_n)$. If the algorithms used to compute that value are accurate enough it can be assumed that the only error committed at this stage is a final rounding of the obtained number (to one word). Then $\tilde{f}(\ldots) = f^*(\ldots)$ and the inherent error appears to be of an order $1/T$ times smaller than the induced error. As a consequence, whenever T can be made very small

$$\pi_{n+1} \gg \rho_{n+1} \tag{3-10}$$

Henrici [1] has already suggested that using "Partial Double Precision" in the integration algorithm could eliminate the induced error and therefore result in substantially decreasing the effect of round off errors. It will be shown that this very selective use of double precision will deeply affect the accumulation of round off errors when small word computers are used. However, before this point can be discussed, some general results about the local and the accumulated round off error have to be introduced.

3.3 The Local Round Off Error

The way in which round off errors are generated in the process of performing a particular arithmetic operation depends greatly upon the implementation of the arithmetic and the type of computer used (there is a difference for example if the computer uses a simple or a double accumulator [2]). However, generally speaking, there are two ways to round a number to a N-bit mantissa floating point expression where the first bit is reserved for the sign [3], [4]:

1) by simple truncation (or chopping), then all bits beyond the N most significant ones are simply dropped
2) by symmetric rounding, then a special algorithm is required: first the (N+1)th bit of the mantissa is added to the Nth one, then only the N most significant bits of the result are retained.

The N-bit mantissa floating point representation of any number y can therefore be written as

$$y^* = y(1 - e) \tag{3-11}$$

where

$$0 \leqslant e < 2^{-N+1}$$

if y is simply truncated to form a N-bit mantissa floating point number, and

$$-2^{-N} \leqslant e < 2^{-N}$$

if y is symmetrically rounded.

The local relative round off error e cannot be known a priori for every number that will be used during the integration process except for its bounds. Therefore, the relative local round off error will be viewed as a random variable with uniform distribution. Moreover, it will be assumed that all those random variables corresponding to successive steps of integration are independent. Then the laws about summing a large number of independent random variables will be applicable.

When no special rounding algorithm is included in the arithmetic, computer will naturally round the results of all operations by ıply truncating them. But, because of the two's complement form ɔd to represent negative numbers, it will actually truncate their solute values [1]. Therefore, in these conditions the only safe /pothesis that can be formulated about the statistical characteristics f the local round off error are

$$|E(e_n)| \leq \mu|x_n| \quad , \qquad \mu = 2^{-(N+1)} \tag{3-12a}$$

$$\text{var}(e_n) = \sigma^2 x_n^2 \quad , \qquad \sigma^2 = 2^{-2N}/12 \tag{3-12b}$$

When an algorithm performing symmetric rounding is included in the arithmetic, it then can be assumed that

$$E(e_n) = 0 \tag{3-13a}$$

$$\text{var}(e_n) = \sigma^2 x_n^2 \quad , \qquad \sigma^2 = 2^{-2N}/12 \tag{3-13b}$$

3.4 The Accumulated Round Off Error

Substracting (4) from (6), and defining

$$r_n = \tilde{x}_n - x_n$$

as the accumulated round off error at the n^{th} step of integration yields

$$r_{n+1} = r_n + T\,[\,f(\tilde{x}_n, \tilde{u}_n) - f(x_n, u_n)\,] + e_{n+1}$$

This equation describes the evolution of the accumulated round off error. It can be approximated by

$$r_{n+1} = r_n + T\,[(\frac{df}{dx})_n r_n + (\frac{df}{du})_n(\tilde{u}_n - u_n)] + e_{n+1} \qquad (3\text{-}14)$$

In order to focus more specifically on the errors introduced by the integration process itself, it will now be assumed that

$$\tilde{u}_n - u_n = 0$$

so that the accumulated round off error becomes

$$r_{n+1} = r_n + T\,g_n r_n + e_{n+1} \qquad (3\text{-}15)$$

where

$$g_n = (\frac{df}{dx})_n$$

Equations (14) and (15) relate the accumulated round off error to the local round off error and to the parameters of the original differential equation. Looking at the expression found for the local round off error (7) as well as at those two last equations, it clearly appears that the accumulated round off error depends on all the aspects of the original simulation problem in a rather compex manner. To approach it mathematically will therefore often be hopeless. However, Henrici [1] has established some theoretical results which will now be introduced without proof.

He has shown that assuming

$$|E(e_n)| \leq \mu p_n \tag{3-16}$$

$$\text{var}(e_n) = \sigma^2 q_n \tag{3-17}$$

where p_n and q_n are two continuous, nonnegative, piecewise smooth functions of $t = nT$, the statistical characteristics of the accumulated round off error are given by

$$|E(r_n)| \leq \frac{\mu}{T}(m_n + O(T)) \tag{3-18}$$

$$\text{var}(r_n) = \frac{\sigma^2}{T}(v_n + O(T)) \tag{3-19}$$

where m_n and v_n are two functions of $t = nT$ obtained by solving the following differential equations

$$m'(t) = g(t)m(t) + p(t) \tag{3-20}$$

$$m(0) = 0$$

and

$$v'(t) = 2g(t)v(t) + q(t) \tag{3-21}$$

$$v(0) = 0$$

where

$$g_n = \left(\frac{df}{dx}\right)_n$$

This result indicates that the expected value and the variance of the accumulated round off error, which will be assumed to have a Gaussian distribution, are directly proportional to the expected value and the variance of the relative local round off error. It also emphasizes that the statistical characteristics of the accumulated round off error will be inversely proportional to the step size of integration T, a fact that apparently prohibits using in the same time short word computers and small step sizes. Partial double precision will change this fact.

Example

Consider a simple first order system with a unit step input

$$x'(t) = -Lx(t) + 1 \qquad (3\text{-}22)$$

$$x(0) = 0$$

The exact solution to this problem is given by

$$x(t) = \frac{1}{L}(1 - e^{-Lt}) \qquad (3\text{-}23)$$

It is represented by a broken line in figure 1, for L = 1 and L = 0.1. If the numerical integration were perfect, its results would correspond to points on those curves. Actually when the simulation is performed with a 9-bit mantissa computer (N = 8), for example, the curves obtained are quite different as shown on the figure. The output of the integrator becomes constant at a level much below the limit toward which it is supposed to tend. This shows that there is a minimum rate of change of the variable to integrate under which the numerical integration becomes inoperative. What happens can be expressed by

$$\tilde{x}_{n+1} = \{\tilde{x}_n + [T\,\tilde{f}(\tilde{x}_n, \tilde{u}_n)]\}^* = \tilde{x}_n$$

or

$$T\,\tilde{f}(\tilde{x}_n, \tilde{u}_n) < 2^{-N}(\text{exponantial part of } x_n)$$

$$x'_n < \frac{2^{-N}}{T}(\text{exponantial part of } x_n)$$

When T and N are small this phenomena may generate very large errors. For example, setting L = 1. in (22) and integrating (22) with a 9-bit mantissa computer will introduce a steady state error of about 100% when T = 1/128 sec. and an error of about 300% when T = 1/512 sec.

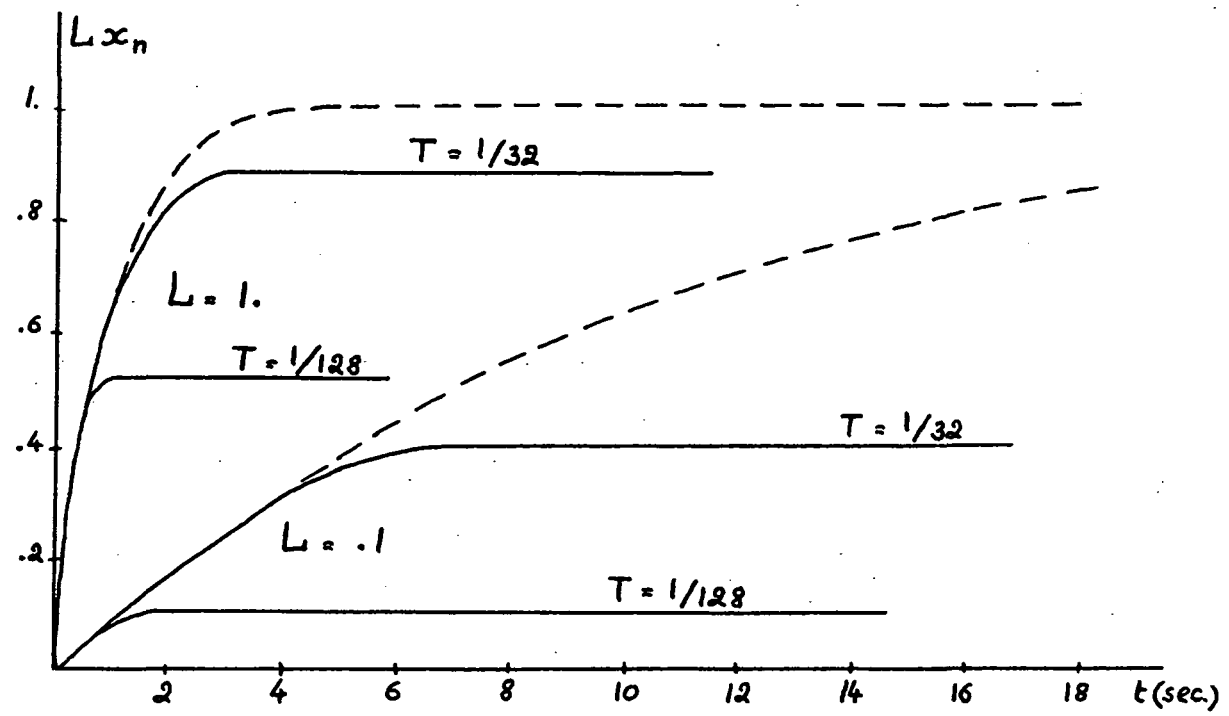

Fig. 3-1. Integration of $x' = -Lx + 1$ with a computer having a 9-bit mantissa (including the sign bit). The exact solution to this differential equation is indicated with a broken line.

3.5 Partial Double Precision

As previously mentioned (10), in most cases of real time simulation the induced error forms the largest part of the local round off error. To eliminate it would therefore result in a substantial gain in accuracy, particularly when short word computers are used. This can precisely be achieved at low cost (a very small increase in computation time) by performing the addition on the right side of (4) in double precision. Henrici [1] refers to this algorithm as "partial double precision".

Using partial double precision, the equation expressing the real results of the numerical integration, taking the effect of rounding into consideration, becomes

$$\tilde{x}_{n+1} = \left\{\tilde{x}_n + T\,\tilde{\tilde{f}}(\tilde{x}_n^*, \tilde{u}_n^*)\right\}^{\#} \qquad (3\text{-}25)$$

where the following definitions apply:

1) $\tilde{x}$ is the double precision computed value of the quantity x
2) x^* is the value of x after it has been rounded to form a single precision quantity
3) $y^{\#}$ is the double precision rounded value of the quantity y
4) $\tilde{f}$ is the single precision computed value of the function f.

It is here emphasized that although x_n will be obtained step after step as a double precision variable, only its single precision value is used to compute the function f(...). Therefore, this last computation which in many instances will be the most time consuming, especially when the function is highly non-linear, will not require any additional time. Then, if it is assumed that the function f(...) can be calculated accurately enough so that the only error degrading its value is its final rounding to a single precision number

$$\tilde{\tilde{f}}(\tilde{x}_n^*, \tilde{u}_n^*) = f^*(\tilde{x}_n^*, \tilde{u}_n^*)$$

and (25) can be somewhat simplified

$$\tilde{x}_{n+1} = \left\{\tilde{x}_n + T\,f^*(\tilde{x}_n^*, \tilde{u}_n^*)\right\}^{\#} \qquad (3\text{-}26)$$

Once again, equation (26) may be rewritten in a more manageable form as

$$\tilde{x}_{n+1} = \tilde{x}_n + T\, f(\tilde{x}_n, \tilde{u}_n) + e_{n+1} \qquad (3\text{-}27)$$

where the local round off error is explicitly expressed. Equation (27) appears to be formally identical to equation (6), however two major differences should be noted:

1) $\tilde{x}_{n+1}$ is now a double precision value, and this alone suggests a substantial gain in accuracy
2) e_{n+1} is very different than in equation (6), and will actually absorb part of the gain.

The local round off error can now be estimated. As before, in order to focus on the errors introduced by the integration process itself, it will be assumed that the forcing term is known with an absolute accuracy. Then

$$\tilde{u}_n = u_n = u_n^*$$

And

$$\begin{aligned} e_{n+1} = {} & \{\tilde{x}_n + T\, f^*(\tilde{x}_n^*, u_n)\}^* - \{x_n + T\, f^*(\tilde{x}_n^*, u_n)\} \\ & + T\, f^*(\tilde{x}_n^*, u_n) - T\, f(\tilde{x}_n^*, u_n) \\ & + T\, f(\tilde{x}_n^*, u_n) - T\, f(\tilde{x}_n, u_n) \end{aligned}$$

$$\begin{aligned} = {} & \{\ldots\}^* - \{\ldots\} + T\,[\, f^*(\ldots) - f(\ldots)\,] \\ & + T g_n(\tilde{x}_n^* - \tilde{x}_n) \end{aligned} \qquad (3\text{-}28)$$

where

$$g_n = \left.\frac{df(\ldots)}{dx}\right|_n$$

Following the same line of reasoning as before, the quantities

$$(\tilde{x}_n^* - \tilde{x}_n)$$

and

$$[\, f^*(\ldots) - f(\ldots)\,]$$

will be viewed as two random processes. Specifically it will be assumed that

$$\left|E(\tilde{x}_n^* - \tilde{x}_n)\right| \leq \mu' |x_n| \tag{3-29a}$$

when numbers are simply truncated, or,

$$E(\tilde{x}_n^* - \tilde{x}_n) = 0 \tag{3-29b}$$

when numbers are symmetrically rounded, and

$$\text{var}(\tilde{x}_n^* - \tilde{x}_n) = \sigma'^2 x_n^2 \tag{3-29c}$$

in both cases. In the same way, it will also be assumed that

$$\left|E[f^*(\ldots) - f(\ldots)]\right| \leq \mu' |f(x_n, u_n)| \tag{3-30a}$$

when numbers are simply truncated, or

$$E[f^*(\ldots) - f(\ldots)] = 0 \tag{3-30b}$$

when numbers are symmetrically rounded, and

$$\text{var}[f^*(\ldots) - f(\ldots)] = \sigma'^2 f^2(x_n, u_n) \tag{3-30c}$$

in both cases. For all those expressions

$$\mu' = 2^{-(N+1)} \tag{3-31a}$$

$$\sigma'^2 = 2^{-(N+1)}/12 \tag{3-31b}$$

The statistical characteristics of the local round off error, when simple truncation is used, can then be found by introducing (29) and (30) into (28), so that

$$\left|E(e_{n+1})\right| \leq \mu |x_n| + T\mu' |f(x_n, u_n)| + T g_n \mu' |x_n| \tag{3-32}$$

where

$$\mu = 2^{-(M+1)}$$

and M is the number of bits in the mantissa (not including the sign bit) of a double precision number. Also, applying the rules regarding the summation of independent random variables, it can be found that

$$\mathrm{var}(e_{n+1}) = \sigma^2 x_n^2 + T^2 \sigma'^2 f^2(x_n, u_n) + T^2 g_n^2 \sigma'^2 x_n^2 \qquad (3\text{-}33)$$

where

$$\sigma^2 = 2^{-2M}/12$$

When symmetric rounding is used for all computations the expected value of the local round off error is of course assumed to be

$$E(e_{n+1}) = 0 \qquad (3\text{-}34)$$

while its variance is still given by (33).

Now, assuming that μ is very small compared to μ' and σ is very small compared to σ', the preceeding expressions can be somewhat simplified. Equation (32) becomes

$$|E(e_{n+1})| \leq T \mu' [|f(x_n, u_n)| + g_n |x_n|] \qquad (3\text{-}35)$$

while equation (33) now reads

$$\mathrm{var}(e_{n+1}) = T^2 \sigma'^2 [f^2(\ldots) + g_n^2 x_n^2] \qquad (3\text{-}36)$$

This last assumption could probably not be made in most cases where a large computer is used but it will be valid in many cases involving short word computers. For example if a 16-bit microcomputer has its word divided into 7 bits for the exponantial part and 9 bits for the mantissa, its double precision mantissa will have 25 bits and the first term in (32) and (33) will be negligible compared to others for most realizable T's.

At this point, Henrici's formulas to evaluate the statistical parameters of the accumulated round off error [see (18) and (19)] can be called upon to notice that because of the factor T present in the expected value of the local round off error (35), when partial double precision is used together with simple truncation, the expected value

of the accumulated round off error will be independent of the step size of integration and much smaller (very roughly of a factor $1/T$) than in the case where partial double precision was not used. In the same way, it appears that because of the factor T^2 present in the expression of the variance of the local round off error, the general effect of the accumulation of round off errors will decrease with the step size of integration when both partial double precision and symmetric rounding are used. Indeed, in this last case the expected value of the accumulated round off error would (at least theoretically) be zero, while its variance would be proportional to T. This is certainly the most interesting consequence of using partial double precision. It should be emphasized however that in order for this to be true the first terms in both (32) and (33) must always (for any n) be negligible compared to the other terms. If, for example, T were very small and as a consequence double precision would not be enough to fulfill this requirement, it could be of interest to program some kind of triple or even quadruple precision algorithm. To do so would not represent a very difficult problem since the corresponding extra-long variables would have to be used in an addition only.

There is a second very positive consequence of using partial double precision : the minimum rate of change of the variable to integrate under which the integrator will not be able to follow properly is now given by

$$x'_n < \frac{2^{-M}}{T} \qquad (3\text{-}37)$$

Since M is considerably smaller than N, (37) represent a substantial improvement compared to (24) when a short word computer is used.

First Example

Consider a simple linear first order system with unit step input to be simulated by a short word computer using partial double precision. The differential equation to integrate is

$$x' = -Lx + 1 , \qquad x(0) = 0 \tag{3-38}$$

Its exact solution is

$$x(t) = \frac{1}{L}(1 - e^{-Lt}) \tag{3-39}$$

The expected value of the local round off error can be found using (35). It should be noted though that because the solution x_n will always (for all n) be positive the absolute value signs in (35) may be removed and the inequality now becomes an equality. The quantities $f(x_n, u_n)$ and x_n cannot be known exactly a priori, however assuming that the results of the numerical integration will closely follow the exact solution, those two quantities can be approximated using the exact theoretical solution (39). Then

$$x_n = \frac{1}{L}(1 - e^{-LnT}) \tag{3-40a}$$

$$f(x_n, u_n) = e^{-LnT} \tag{3-40b}$$

$$g_n = -L \tag{3-40c}$$

Therefore, (35) gives

$$E(e_{n+1}) = -T\mu'[2e^{-LnT} - 1]$$

or,

$$E(e(t)) = -T\mu'[2e^{-Lt} - 1] \tag{3-41}$$

The expected value of the accumulated round off error can now be estimated using (18) and (20). Equation (20) becomes

$$m'(t) = -Lm(t) - T[2e^{-Lt} - 1]$$

$$m(0) = 0$$

Its solution is

$$m(t) = \frac{T}{L}[1 - (1 + 2Lt)e^{-Lt}] \qquad (3-42)$$

It then follows that

$$E(r_n) = \frac{\mu'}{L}[1 - (1 + 2LnT)e^{-LnT}] \qquad (3-43)$$

In the same way, the variance of the local round off error can be approximated by

$$var(e_{n+1}) = T^2\sigma'^2[e^{-2LnT} + (1 - e^{-LnT})^2] \qquad (3-44)$$

And the variance of the accumulated round off error can be estimated using (19) and (20); it has been found to be

$$var(r_n) = \frac{T\sigma'^2}{2L}[1 - 4e^{-LnT} + (3 + 4LnT)e^{-2LnT}] \qquad (3-45)$$

The next two figures represent the evolution of the expected value of the relative accumulated round off error, that is $E(r_n)/x_n$, and of the variance of the relative accumulated round off error, that is $var(r_n)/x_n$ calculated according to (43) and (45).

Experimentation relatively to this example has been conducted in the following way. Equation (38) has been simulated with two parallel integrations. For the first simulation the full length of the mantissa of the computer (a HP2100 using a 32-bit word with a 23-bit mantissa) was used. For the second integration only 9 bits of the mantissa were retained for calculations (that is the mantissa was truncated or rounded to a 9-bit expression after each arithmetic operation). However, in this second computation the full length of the mantissa was used for the "double precision" variables. The accumulated round off error corresponding to a 9-bit mantissa was then estimated as the difference between the results of those two integrations. Furthermore, in order to obtain an average value and an estimated variance, for each set of parameters, equation (38) was simulated 21 times giving for each pass a slightly different amplitude to the forcing term (the 21 different amplitudes were evenly spread between 0.95 and 1.05). Some of the typical curves obtained in the course of this experimentation are given following the figures representing the theoretical curves.

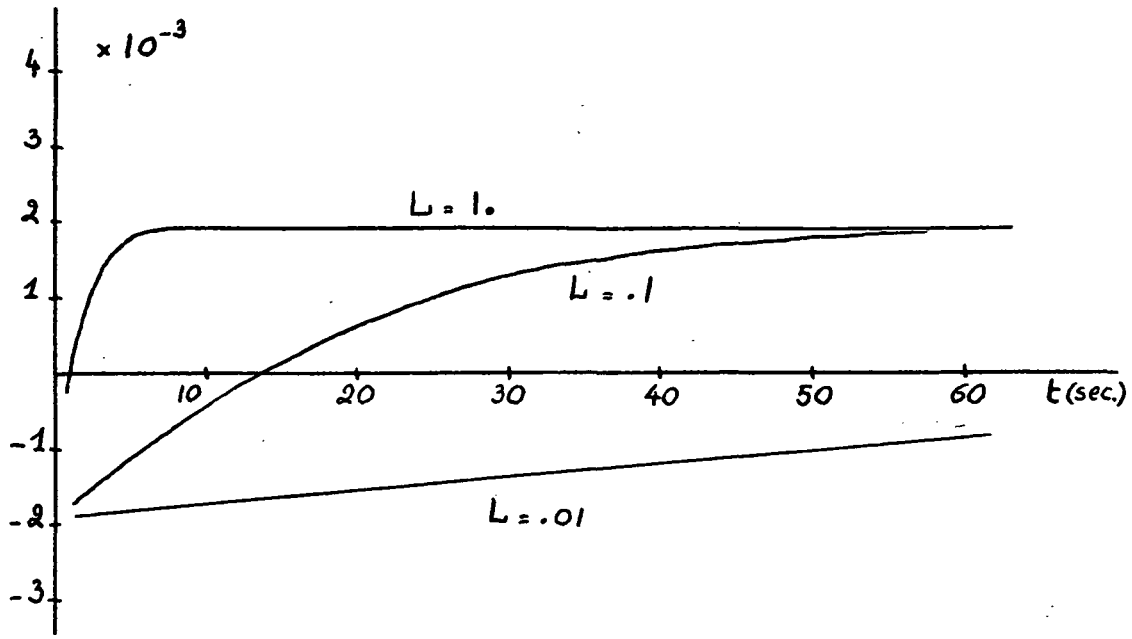

Fig. 3-2. Expected value of the relative accumulated round off error for the first example. Theoretical curve corresponding to equation (43).

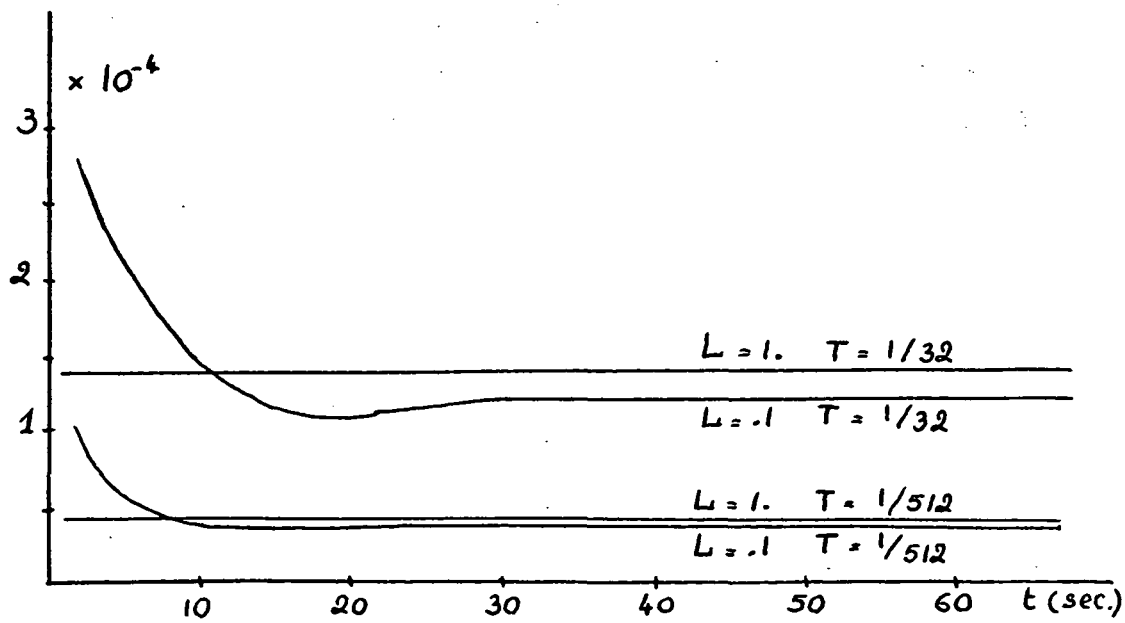

Fig. 3-3. Standard deviation of the relative accumulated round off error for the first example. Theoretical curve corresponding to equation (45).

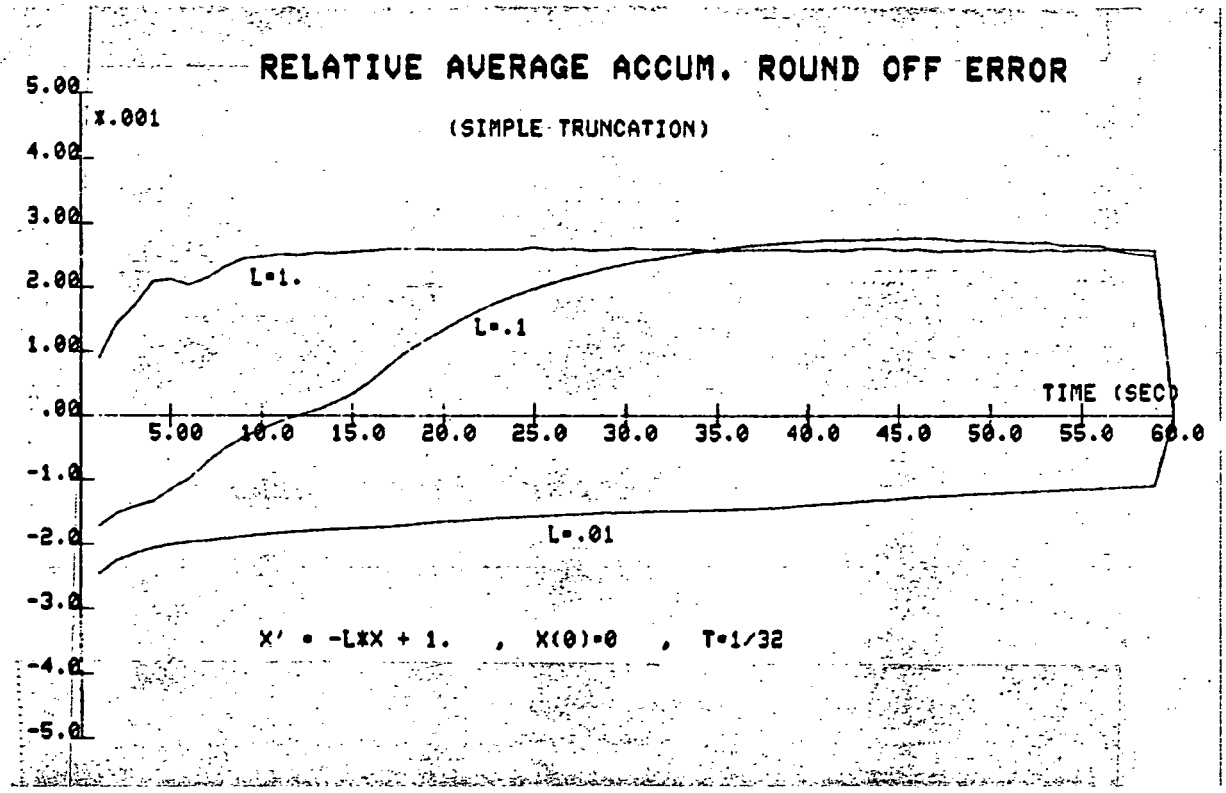

Fig. 3-4. Relative average accumulated round off error corresonding to a real simulation of example 1 (9-bit mantissa).

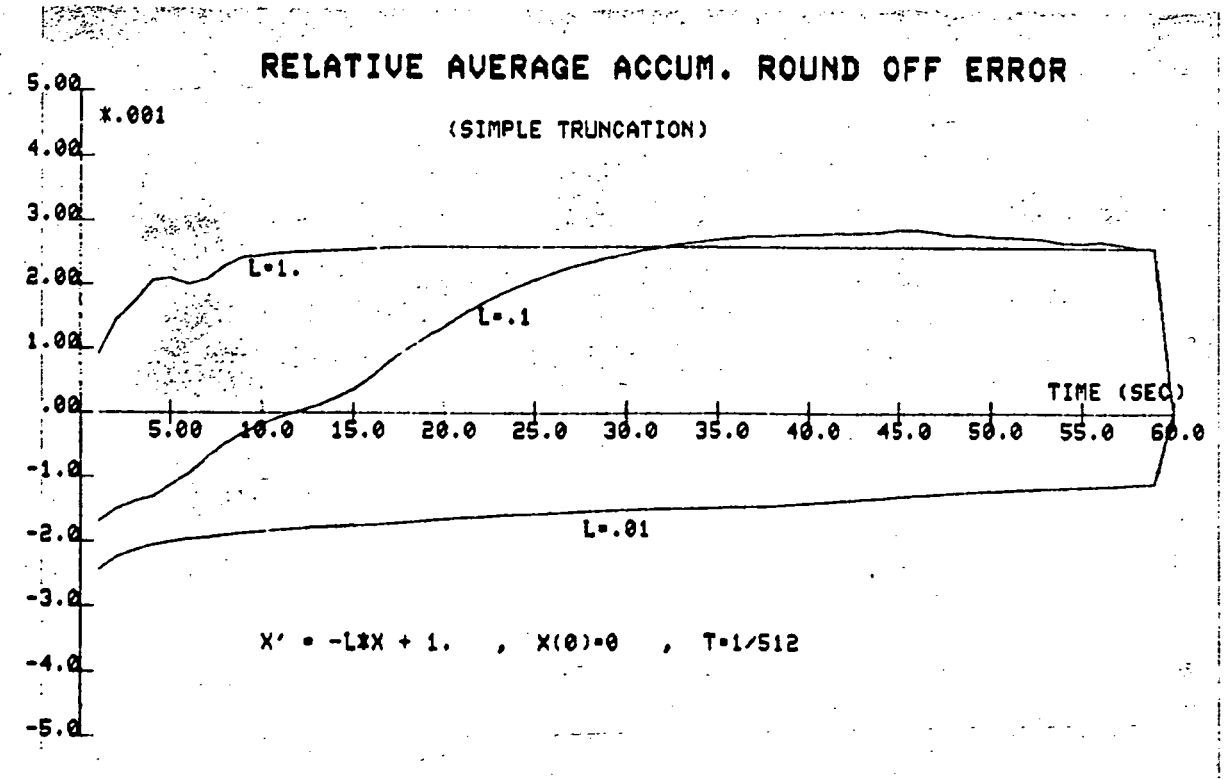

Fig. 3-5. Relative average accumulated round off error corresponding to a real simulation of example 1 (9-bit mantissa).

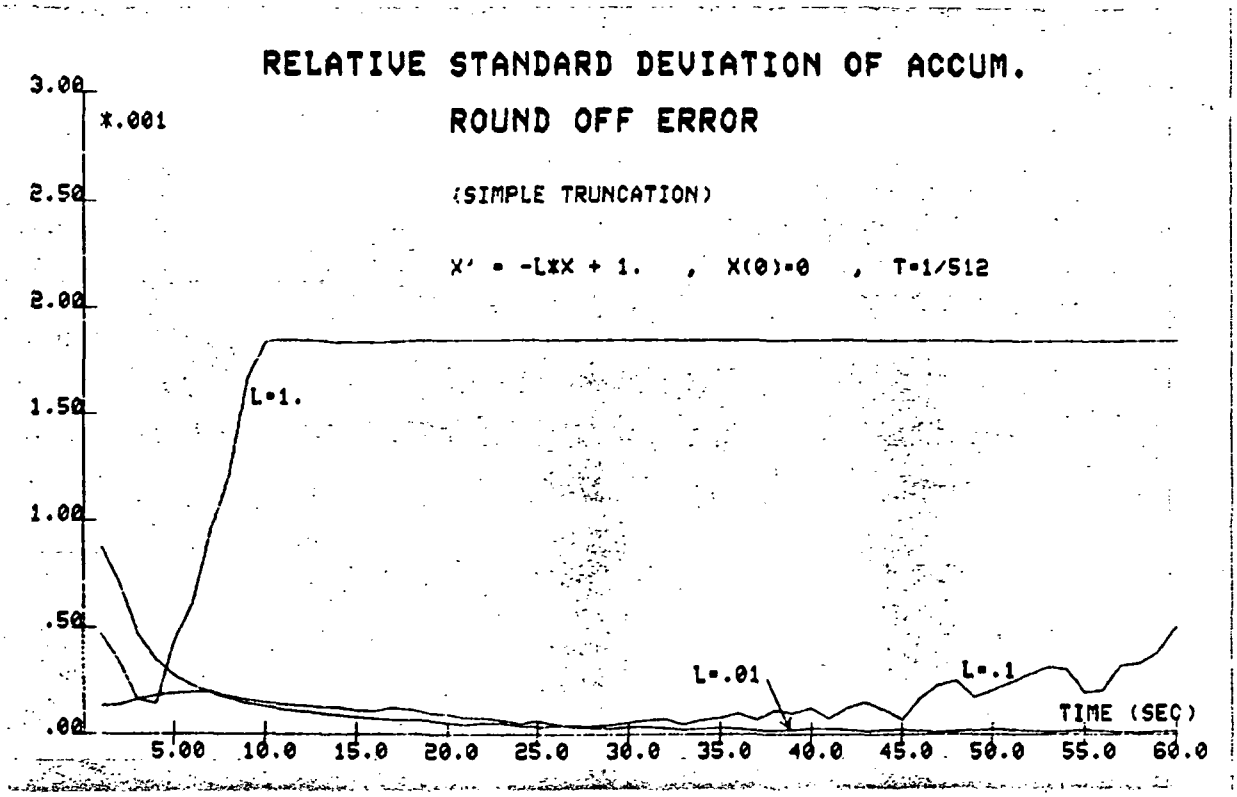

Fig. 3-6. Relative standard deviation of the accumulated round off error corresponding to problem 1 (9-bit mantissa).

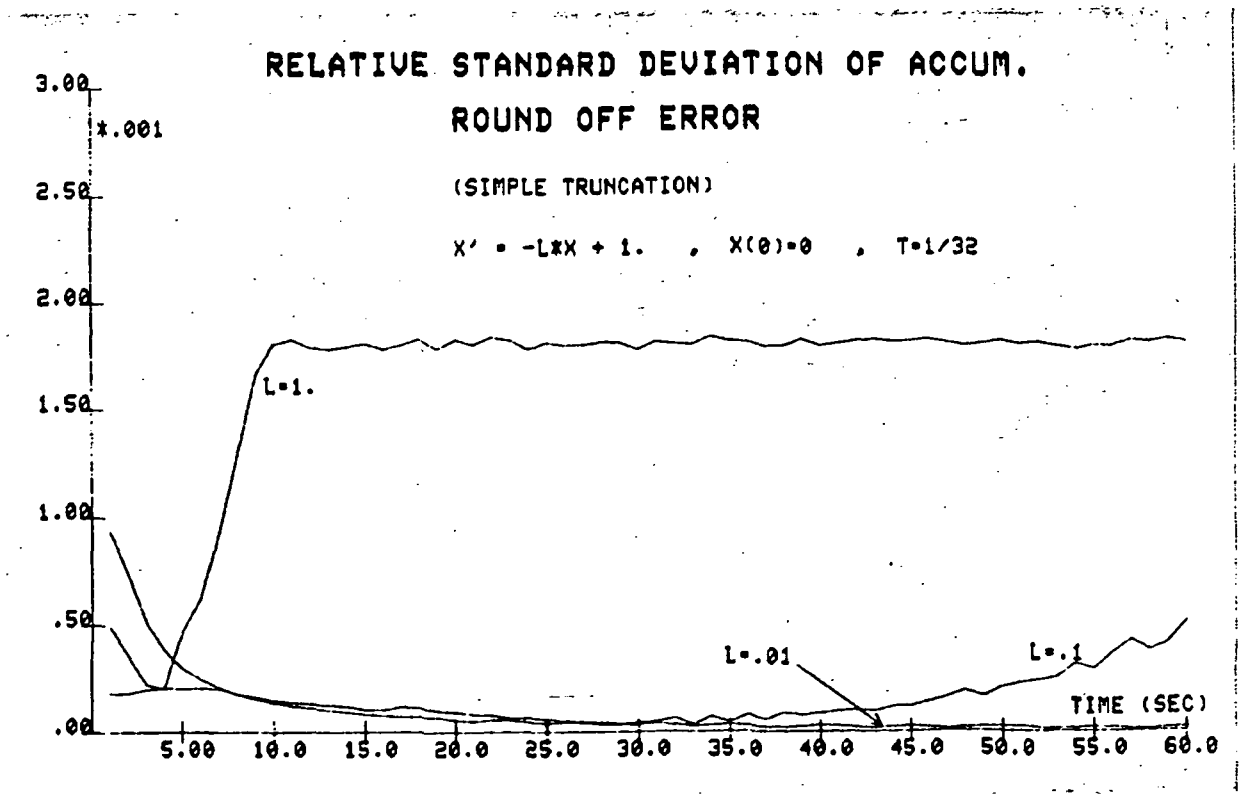

Fig. 3-7. Relative standard deviation of the accumulated round off error corresponding to problem 1 (9-bit mantissa).

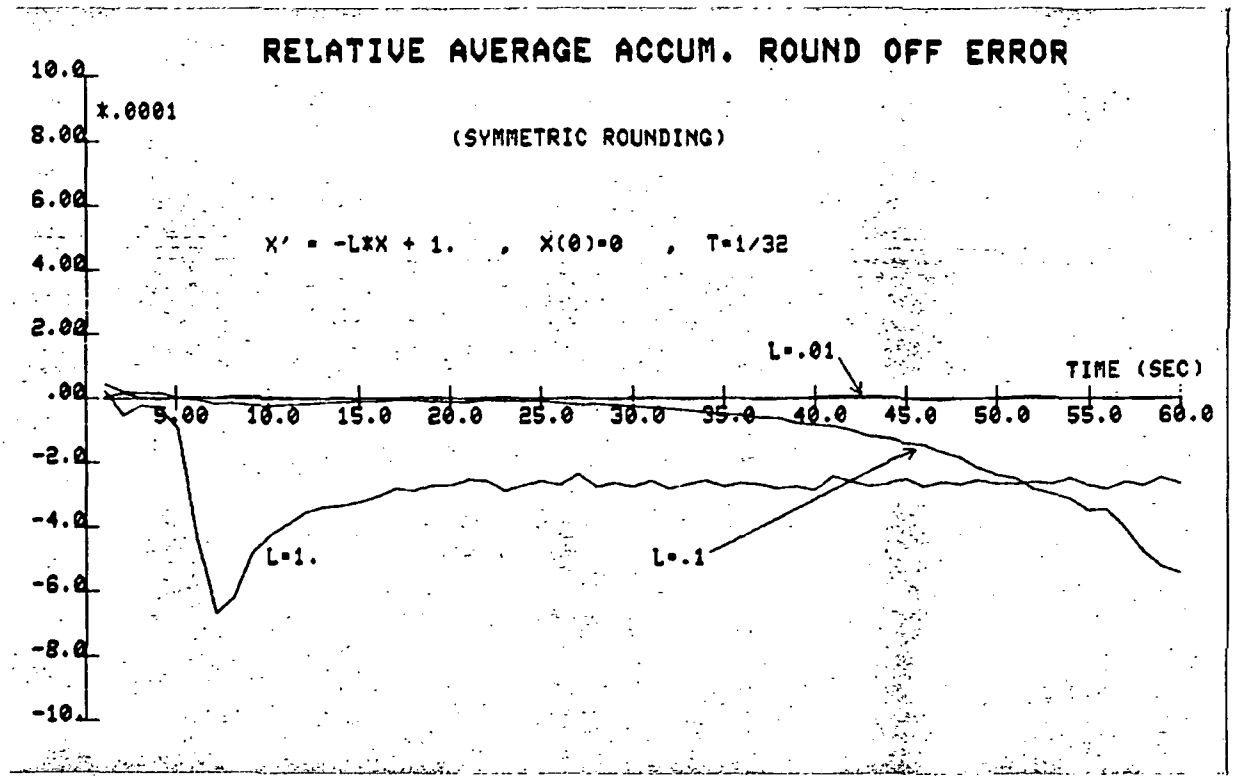

Fig. 3-8. Relative average accumulated round off error corresponding to example 1 (9-bit mantissa).

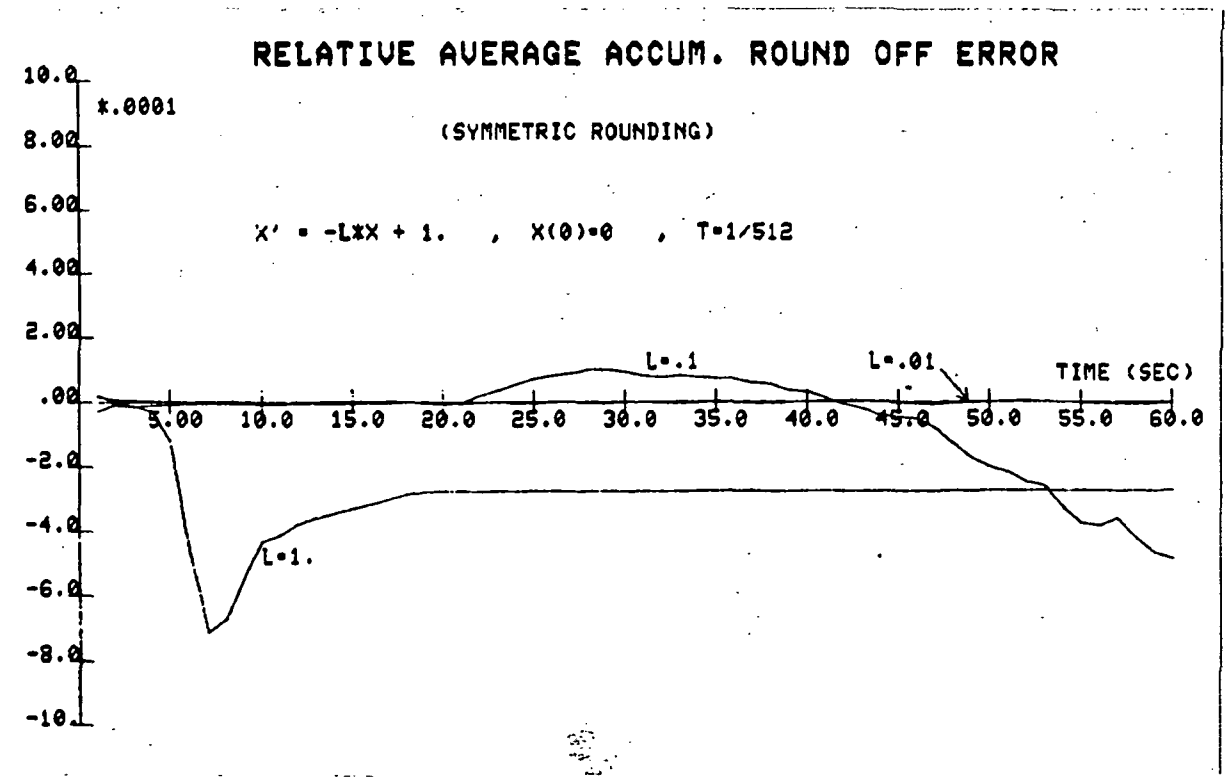

Fig. 3-9. Relative average accumulated round off error corresponding to example 1 (9-bit mantissa).

Second example

Consider now a linear first order system with a sinusoidal input to be simulated using a short word computer and partial double precision. The steady state operation will be assumed in order to examine whether the accumulated round off error will also reach a steady state type of evolution, or on the contrary will grow without bounds. The differential equation to be integrated is

$$x' = -Lx + U \sin wt \tag{3-46}$$

Its steady state solution is

$$x = A \sin(wt + \varphi) \tag{3-47a}$$

where

$$\varphi = \arctan(-w/L) \tag{3-27b}$$

$$A = U/(L\cos\varphi - w\sin\varphi) \tag{3-27c}$$

Using (35), the local round off error is found to be

$$\left|E(e(t))\right| \leqslant T\mu' [\, \left|-LA \sin(wt + \varphi) + U \sin wt\right| - LA\left|\sin(wt + \varphi)\right| \,]$$

or, simplifying,

$$\left|E(e(t))\right| \leqslant UT\mu' \left|\sin wt\right| \tag{3-48}$$

An approximation for (48) can be obtained by the first terms of its Fourier series. Then,

$$\left|E(e(t))\right| \leqslant UT\mu' [\, \frac{2}{\pi} - \frac{4}{\pi}\cos 2wt \,] \tag{3-49}$$

Now the contribution of the two terms on the right side of (49) to the accumulated round off error can be examined separately. The constant term gives, using (20),

$$m_o(t) = \frac{2UT}{L}(1 - e^{-Lt})$$

Therefore (18) becomes

$$\left|E(r_n)\right|_{av} \leqslant \frac{\mu' 2U}{\pi L}(1 - e^{-Lt}) \qquad (3\text{-}50)$$

and for t large enough

$$\left|E(r_n)\right|_{av} \leqslant \frac{\mu' 2U}{\pi L} \qquad (3\text{-}51)$$

The cosine term in (49) gives, using (20),

$$m_c(t) = B\cos(2wt + \Psi) \qquad (3\text{-}51a)$$

where

$$\Psi = \arctan(-2w/L) \qquad (3\text{-}51b)$$

$$B = 4U/\pi(L\cos\Psi - 2w\sin\Psi) \qquad (3\text{-}51c)$$

Formula (51) shows that the accumulated round off error will indeed reach a steady state type of evolution, except when $L = 0$, then it will grow without bounds. This result seems to emphasize very strongly the importance of the ratio U/L which will determine the value of the DC term of the expected value of the accumulated round off error. Formula (52) indicates that this last quantity will oscillate around its DC term, but also that the amplitude of the oscillation will be limited. Indeed, (52c) shows that

- If $w \ll L$, then $B \simeq 2\left|E(r_n)\right|_{av}$
- If $w \simeq L/2$ $B \simeq \left|E(r_n)\right|_{av}$
- If $w \gg L/2$ $B \ll \left|E(r_n)\right|_{av}$

The variance of the local round off error can be found for this example using (36).

$$\mathrm{var}(e(t)) = T^2\sigma'^2[(-LA\sin(wt + \varphi) + U\sin wt)^2 + L^2A^2\sin^2(wt + \varphi)]$$

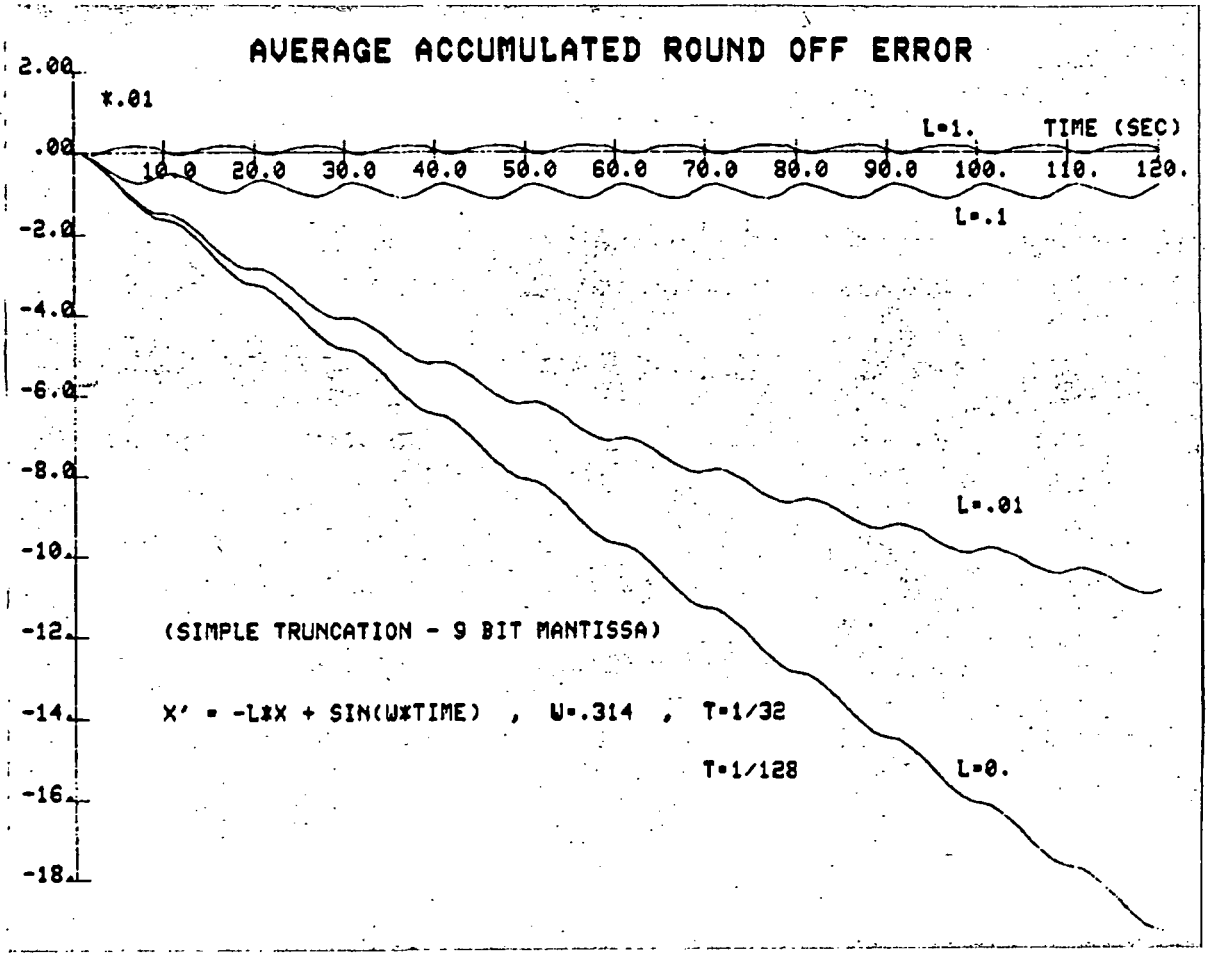

Fig. 3-10. Average accumulated round off error corresponding to the second example. Two other similar simulations performed with w=1. and w=.1 have yielded the same curves except for the ripple.

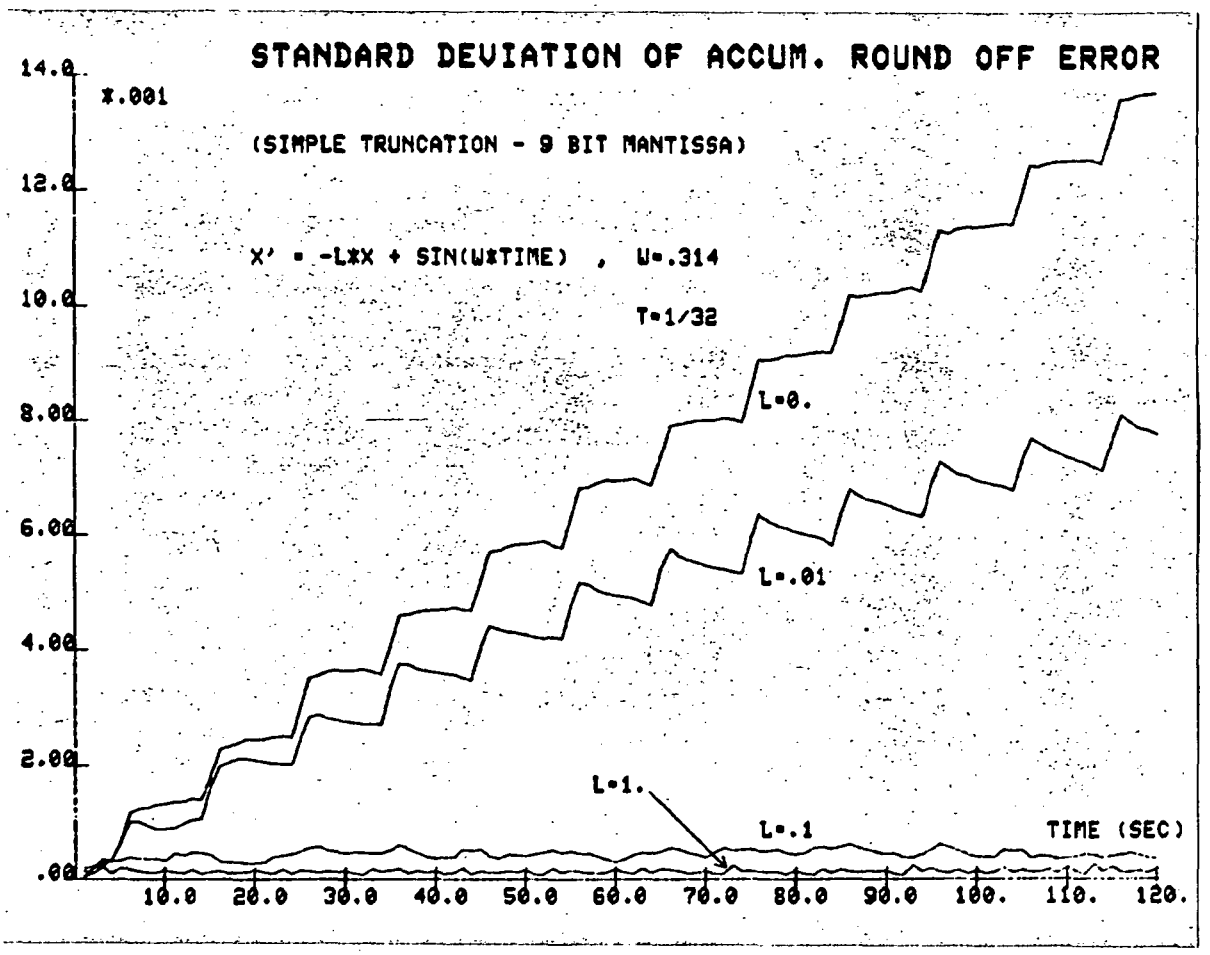

Fig. 3-11. Standard deviation of the accumulated round off error corresponding to example 2. Two similar simulations with w=1. and w=.1 have yielded the same curves except for the ripple.

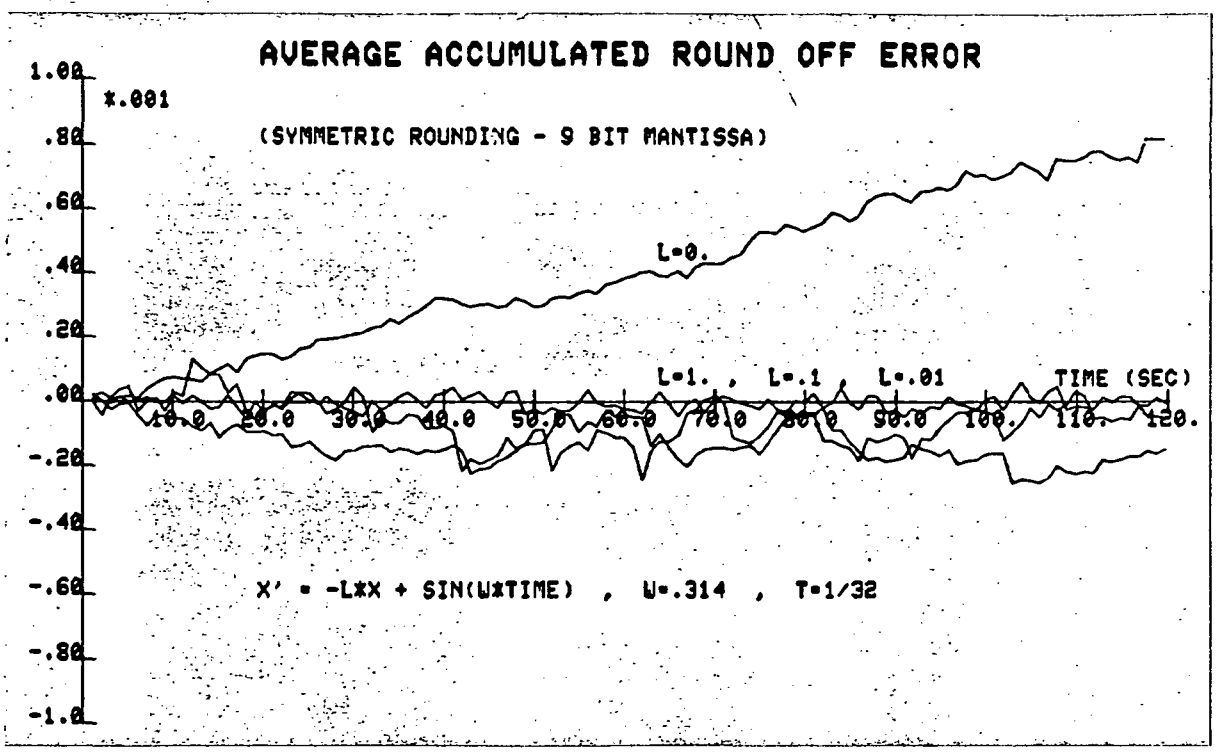

Fig. 3-12. Simulation corresponding to the second example.

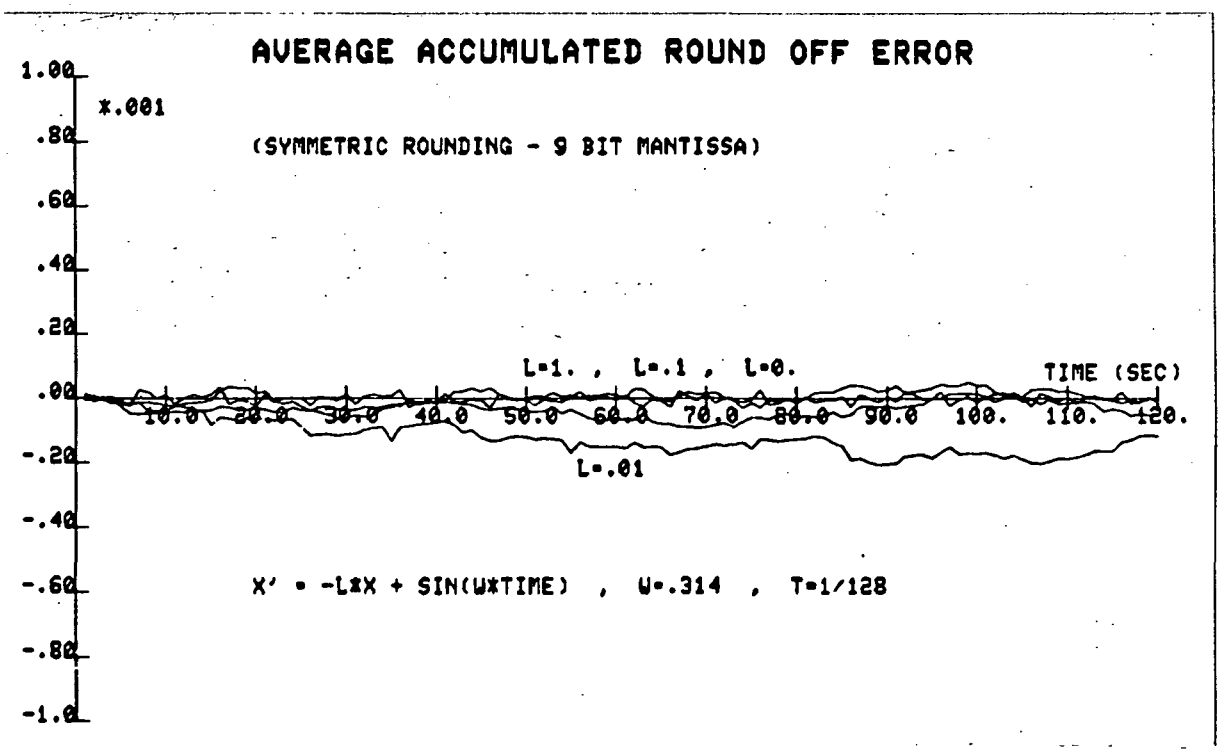

Fig. 3-13. Simulation corresponding to the second example.

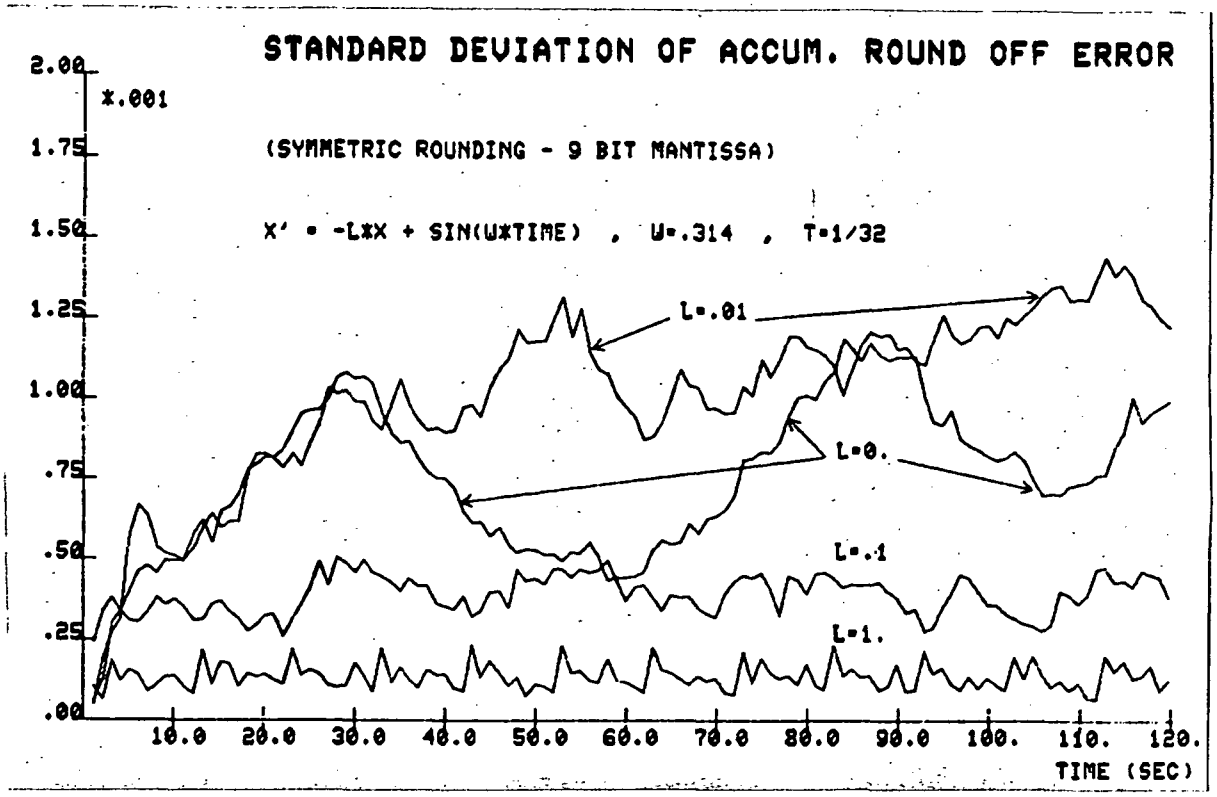

Fig. 3-14. Simulation corresponding to the second example.

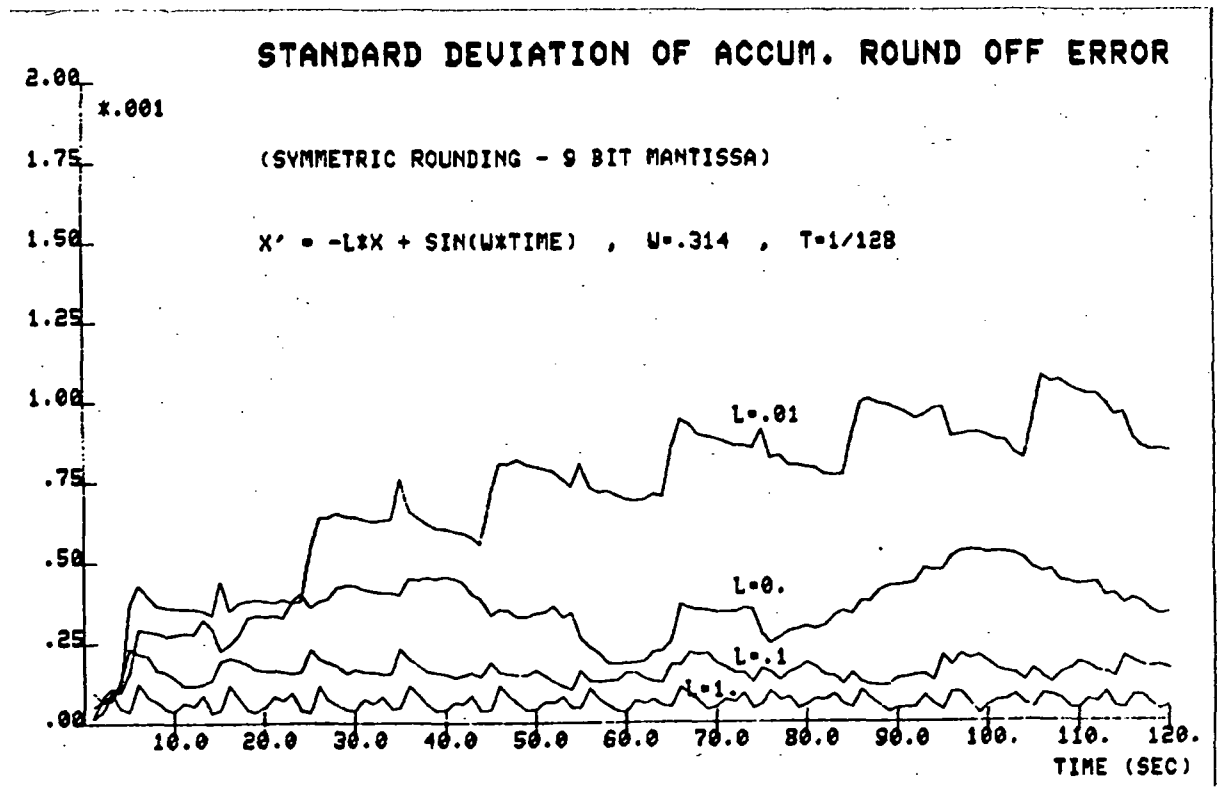

Fig. 3-15. Simulation corresponding to the second example.

or, simplifying,

$$\mathrm{var}(e(t)) \leqslant T^2\sigma'^2(U^2 + 2L^2A^2 + 2LA) \tag{3-52}$$

Then, the accumulated round off error is given by (21) and (19). It was found that

$$v(t) \leqslant \frac{T^2C}{2L}(1 - e^{-2Lt}) \tag{3-53}$$

where

$$C = (U^2 + 2L^2A^2 + 2LA) \tag{3-54}$$

From (53) it can be deduced that the variance $\mathrm{var}(r_n)$ will be bounded. Indeed, (19) gives, for t large enough,

$$\mathrm{var}(r_n)_m \leqslant \frac{\sigma'^2TC}{2L} \tag{3-55}$$

Experimentation regarding this example has been conducted following a procedure similar to the one developed for the first example. Some of the most typical curves obtained are given in the following figures. It results from the various simulations performed that when numbers are simply truncated formulas (50) and (52) give a quite good description of the expected value of the accumulated round off error. It is clear that this quantity can become very large when L is very small. Simple truncation is therefore not advisable with short word computers. When symmetric rounding is used, the expected value of the accumulated round off error is very small, near zero, as anticipated. Then, the most important characteristic of the accumulated round off error becomes its variance or standard deviation. Curves relative to this quantity are also given. They reveal that the variance of the accumulated round off error is badly described by the various expressions established so far. No satisfactory explanation has been found regarding this fact. However, the size of the standard deviation is a comforting factor which sustains the hope of being able to use short word length computers for numerical integration.

3.6 Effect of The Limited Accuracy of The Forcing Function

In many cases of real time simulations, the inputs of the emulated dynamic system will be obtained from analog to digital converters. The accuracy of the corresponding forcing functions will therefore be limited by the number of bits given by the conversion. It can easily be shown that the accumulation of round off errors degrading the results of an integration will, in such a case, be dominated by the effects of the rounding of the forcing term, provided the computer uses a mantissa somewhat longer than the word of the A/D.

For simplicity, the function $f(x_n, u_n)$ will be assumed linear. It can therefore be written as

$$f(x_n, u_n) = - L_n x_n + u_n \tag{3-56}$$

where L_n may be a function of n and where u_n represent the forcing function. The equation corresponding to the evolution of the accumulated round off error (14) now becomes

$$r_{n+1} = r_n + T[- L_n r_n + \Delta u_n] + e_{n+1} \tag{3-57}$$

where

$$\Delta u_n = \tilde{u}_n - u_n \tag{3-58}$$

Equation (57) can be approximated by

$$r'_n = - L_n r_n + T\Delta u_n + e_{n+1} \tag{3-59}$$

The term e_{n+1} is obtained as before from (27) and (28). And, assuming an efficient use of partial double precision, it is found that

$$|E(e_{n+1})| \leqslant T u' [|f(x_n, u_n)| - L_n |x_n|] + T \eta |u_n| \tag{3-60}$$

where η is the expectation of Δu_n. Also,

$$\text{var}(e_{n+1}) = T^2 \sigma'^2 [f^2(\ldots) + L_n^2 x_n^2] + T^2 \gamma^2 u_n^2 \tag{3-61}$$

where γ^2 is the variance of Δu_n. Since the function f(...) has been assumed linear, these last equations can be simplified. Thus,

$$|E(e_{n+1})| \leq T\mu' |u_n| + T\eta |u_n| \tag{3-63}$$

and

$$\text{var}(e_{n+1}) = T^2\sigma'^2 [\, 2L_n^2 x_n^2 - 2L_n x_n u_n + u_n^2 \,] + T^2\gamma^2 u_n^2 \tag{3-64}$$

The expectation and variance of the term Δu_n viewed as a random variable are known since u_n is given by an A/D. They are, assuming that the A/D gives K bits with symmetric rounding,

$$\eta = 0 \tag{3-65a}$$

$$\gamma^2 = \frac{2^{-2K}}{12} \tag{3-65b}$$

If the computer performing the integration uses symmetric rounding and has a mantissa a few bits longer than the word of the A/D, the first terms in (63) and (64) will be negligible compared to the second ones, so that then

$$E(e_{n+1}) = 0 \tag{3-66a}$$

$$\text{var}(e_{n+1}) \cong T^2\gamma^2 u_n^2 \tag{3-66b}$$

These last expressions combined with (59) show that the accumulated round off error at the output of the integrator will depend on the rounding of the input only. As a consequence, if for example the input (or forcing term) is given by a 8-bit A/D, a computer with a 12-bit mantissa will perform as well as a computer with a 40-bit mantissa.

Experimentation on this point has been conducted following a similar procedure as for the preceeding simulations. The results of a first integration performed with a 23-bit mantissa were compared to those obtained in a second inte gration performed with a mantissa reduced to 9 bits by symmetric rounding. The same simulation has been then repeated with, for the second integration, a mantissa reduced to 12 bits by symmetric rounding. The average value and standard deviation of the difference are given in the following figures.

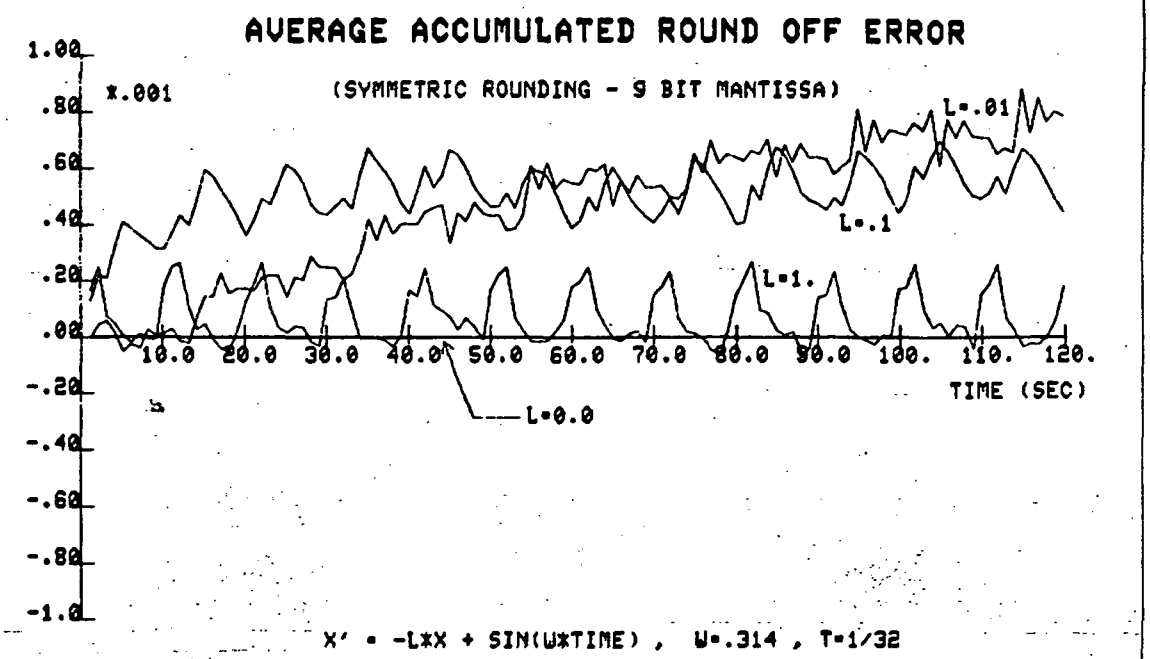

Fig. 3-16.

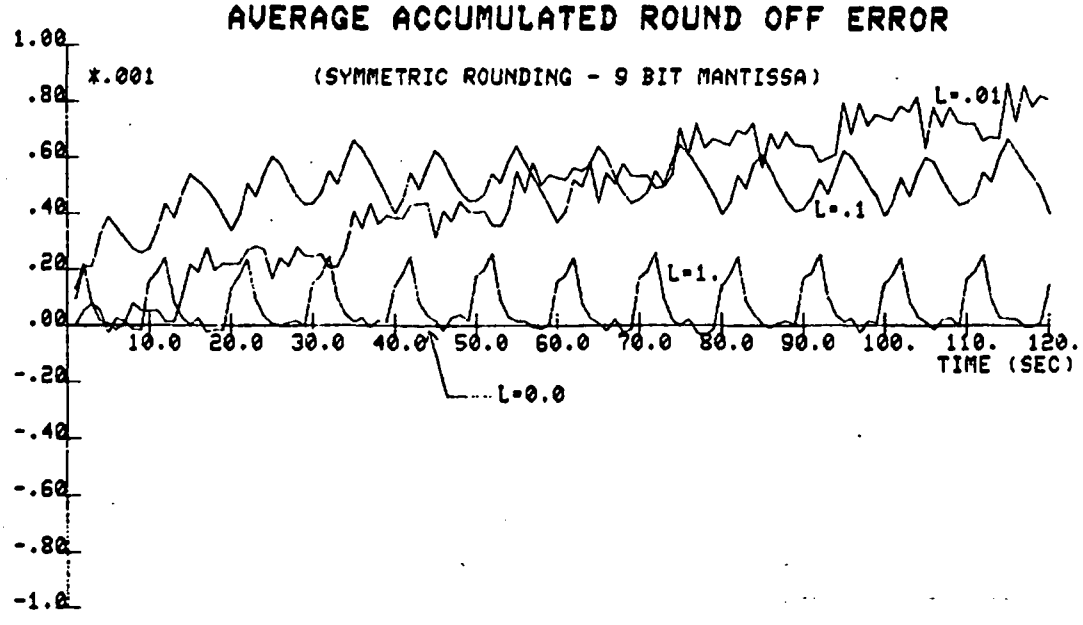

Fig. 3-17.

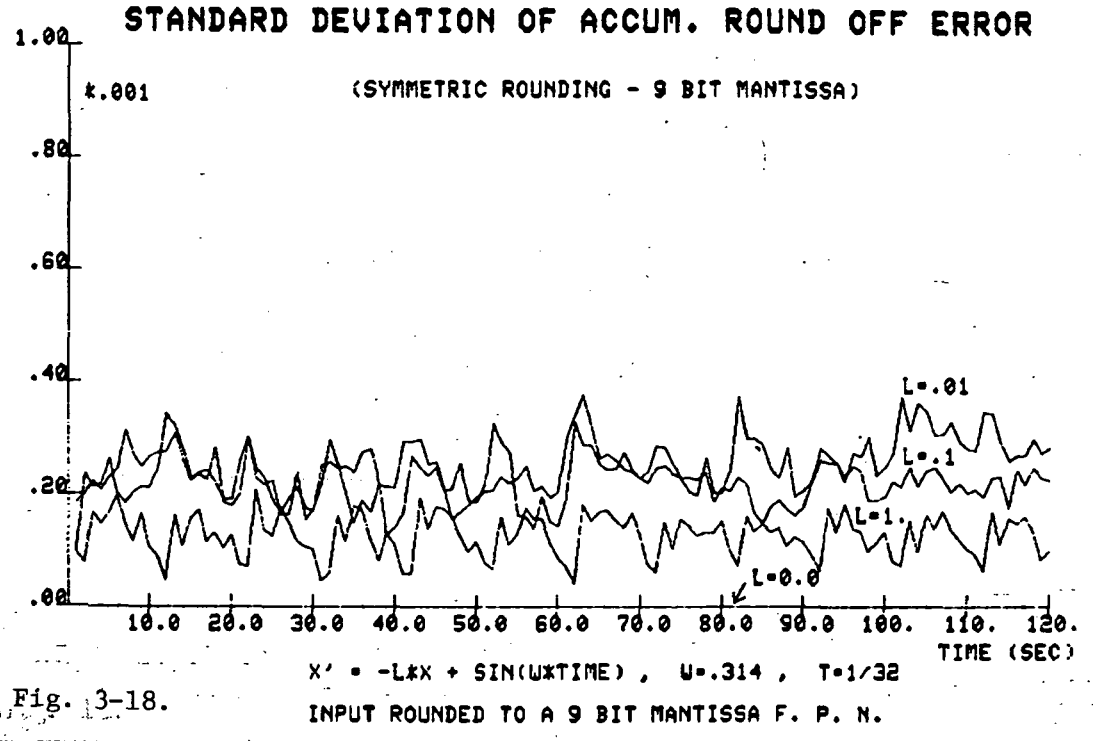

Fig. 3-18.

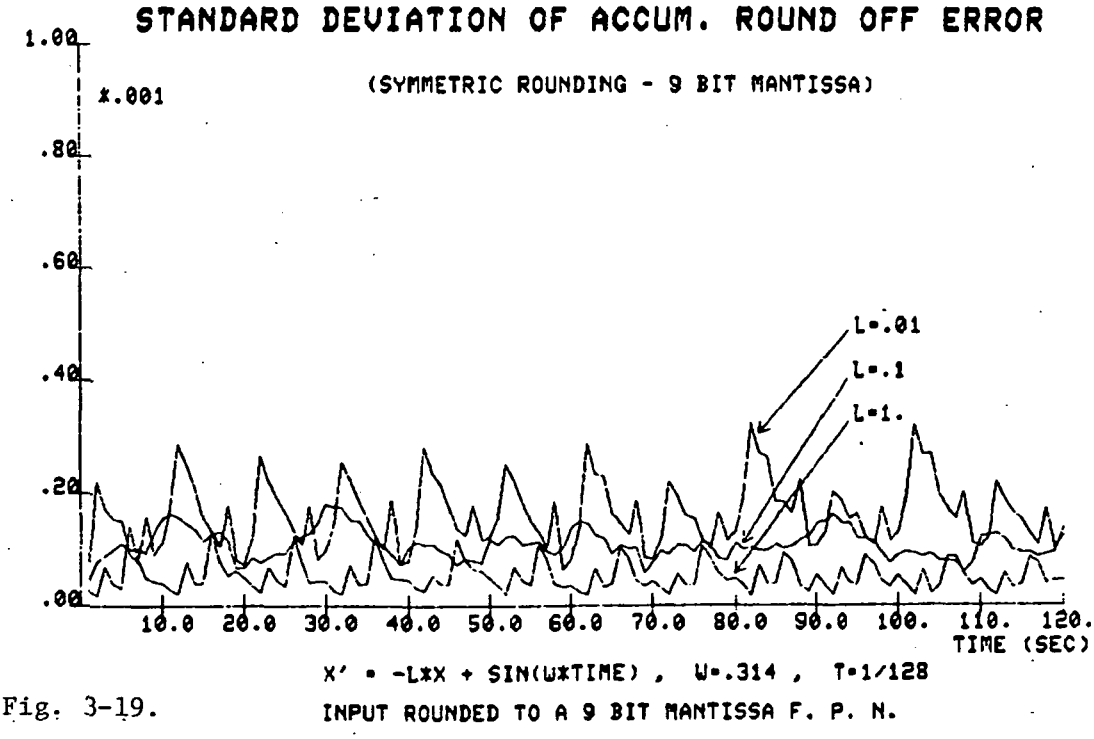

Fig. 3-19.

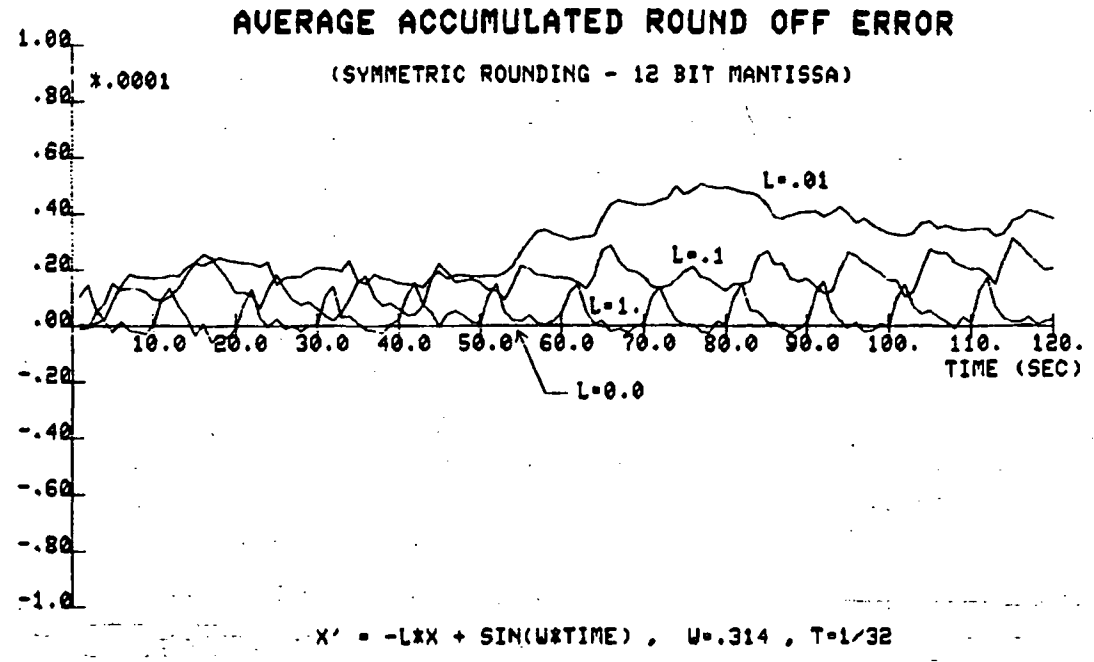

Fig. 3-20.

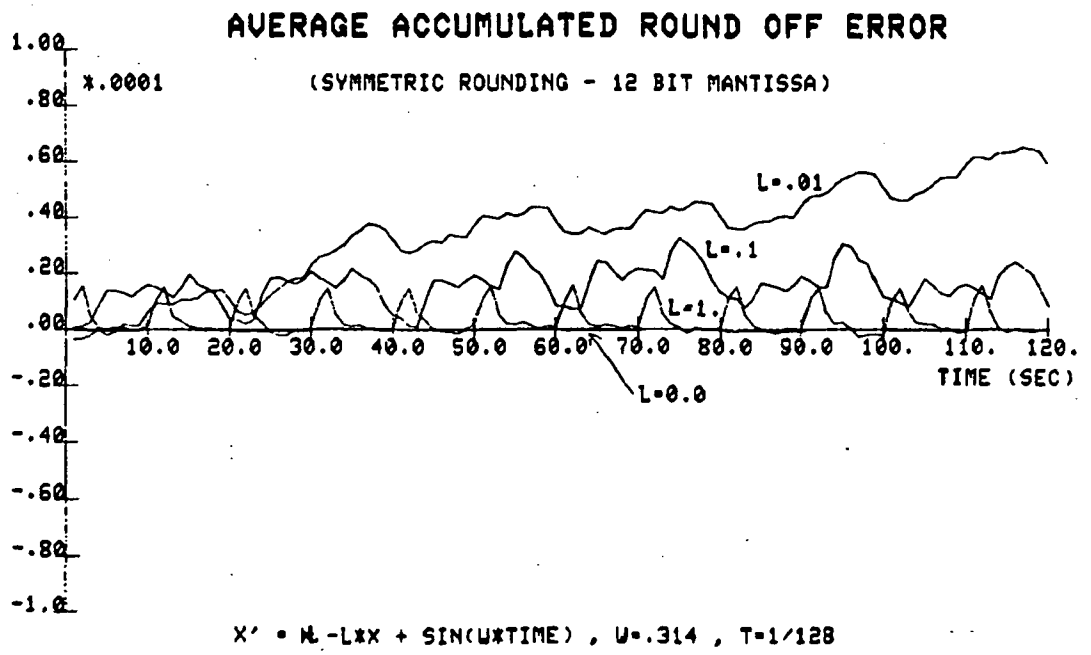

Fig. 3-21.

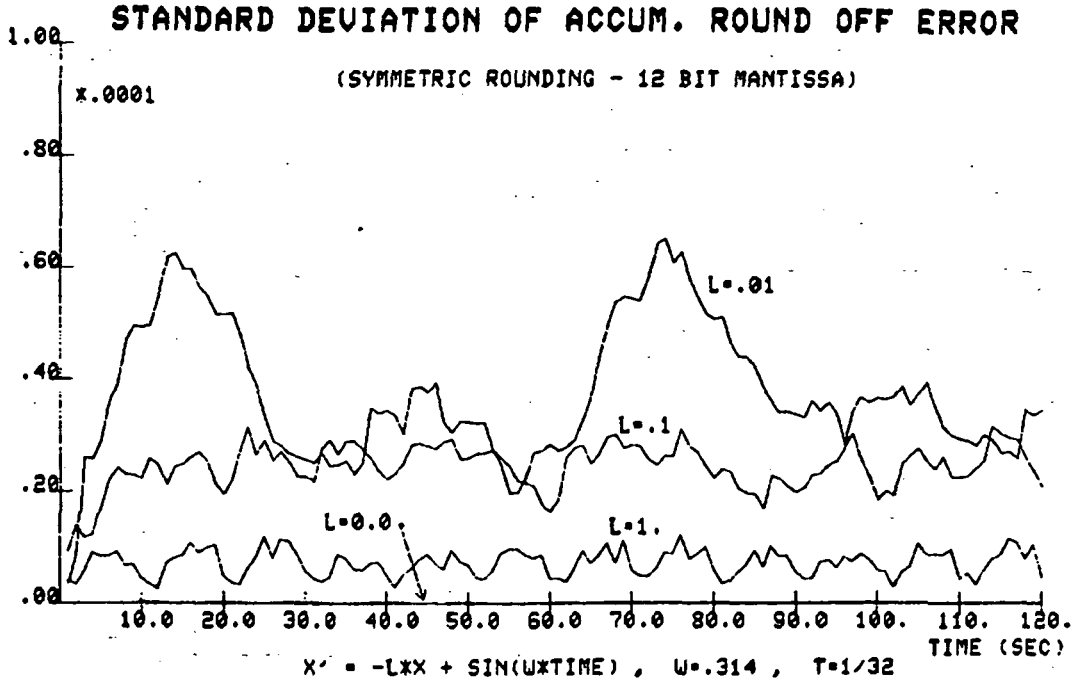

Fig. 3-22.

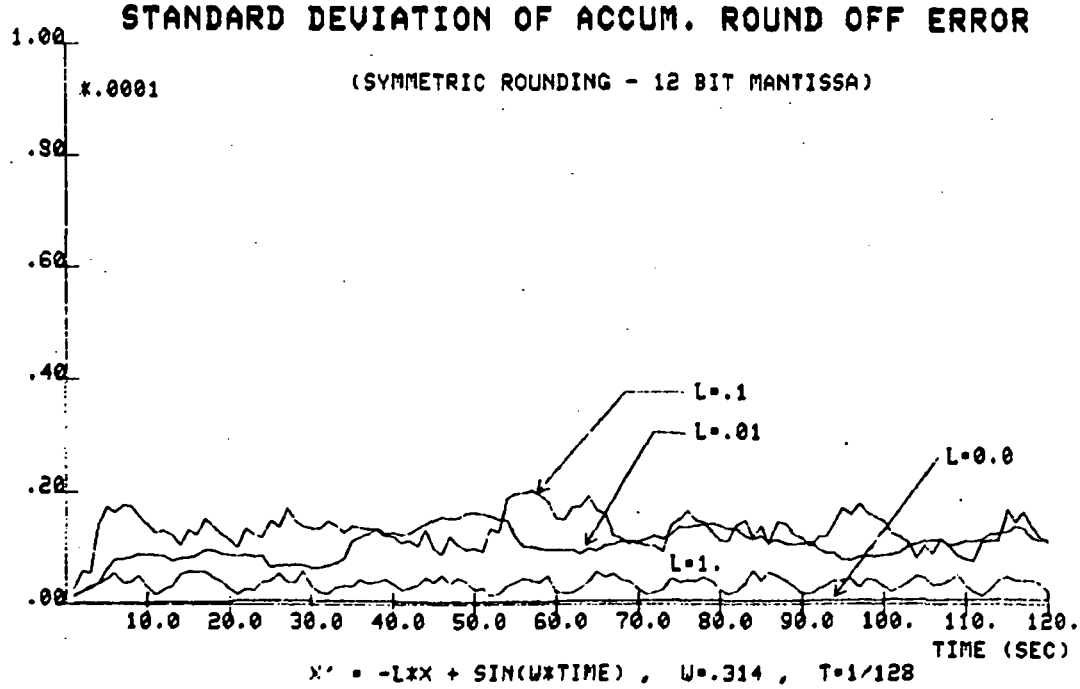

Fig. 3-23.

3.7 Integration of a System of Differential Equations

As mentioned in the introduction, integrating a system of simultaneous differential equations can be done with parallel operations by assigning a separate integrator to each individual equation. After each step of integration the results of those independent computations will be exchanged among the integrators for the updating of the various forcing terms. As a consequence, round off errors will propagate from differential equation to differential equation and may become very large. To study their accumulation mathematically will in many cases be hopeless, especially if the equations are non-linear. Then, simulation will be the only way to obtain an evaluation of their effect. However, the observations made in the preceeding sections apply to integrating systems of equations as well as single equations. Therefore it can be said that if partial double precision and symmetric rounding are used, and if the coefficients L in the equations are not all very small, the accumulated round off error will remain small.

Example

The second order differential equation

$$x'' + 2\zeta w_c x' + w_c^2 x = u \qquad (3\text{-}67)$$

has been simulated as a system of two simultaneous equations

$$\begin{aligned} x' &= y \\ y' &= -w_c^2 x - 2\zeta w_c y + u \end{aligned} \qquad (3\text{-}68)$$

where the forcing term u was a sinusoid of unit amplitude. As for the previous experimentations the integration has been performed twice : once using the full length of the mantissa (23 bits), then reducing the mantissa to 9 bits by symmetric rounding. The average value and the standard deviation of the corresponding accumulated round off error has been calculated and is shown in the following figures.

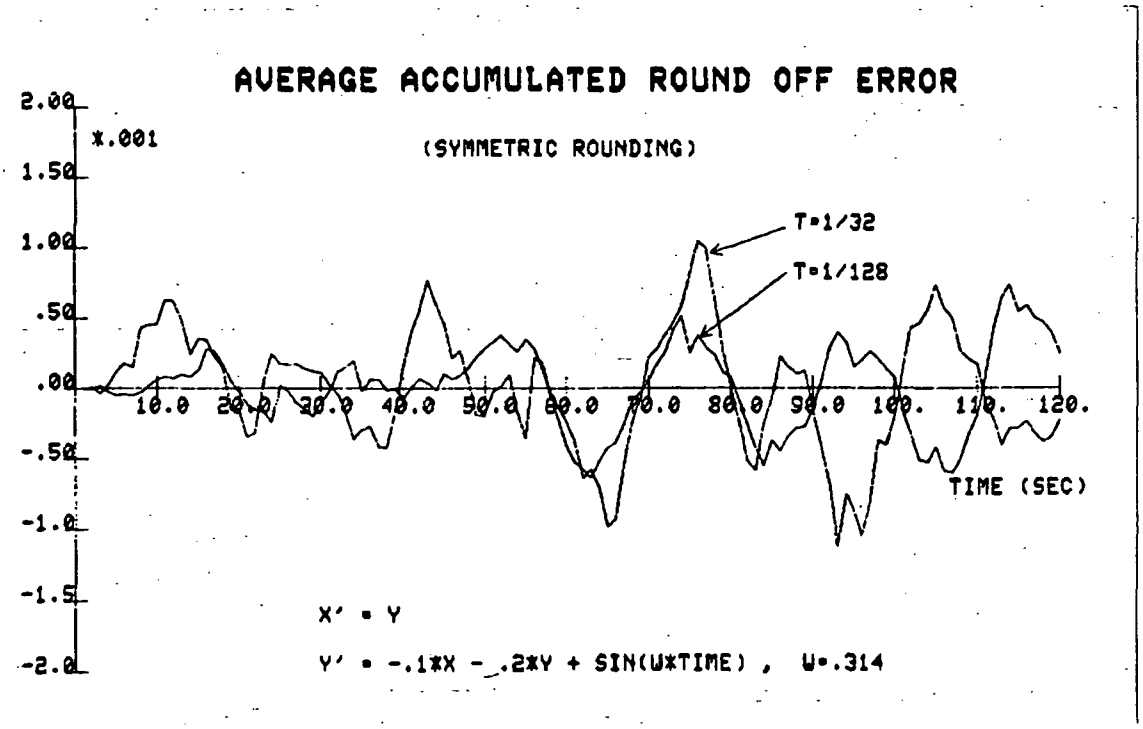

Fig. 3-24. Simulation corresponding to the example on page 36.

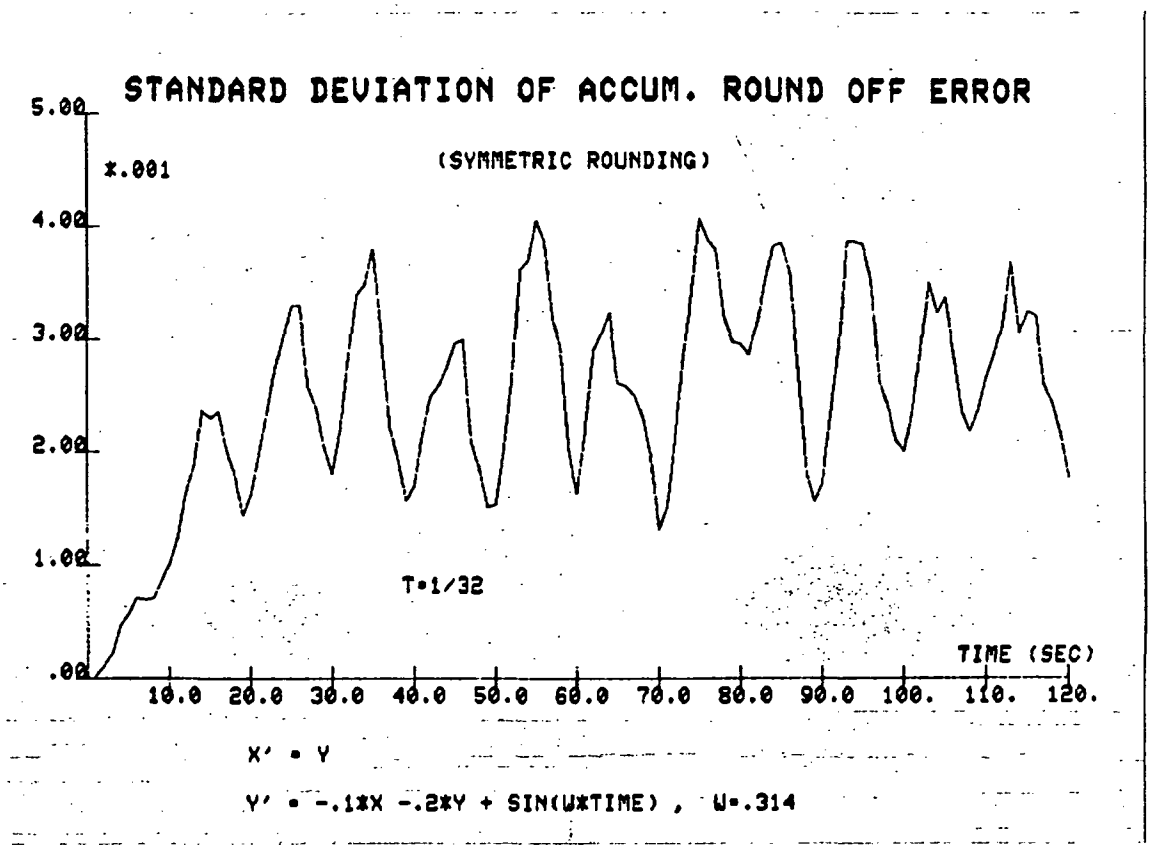

Fig. 3-25. Simulation of a system of second order.

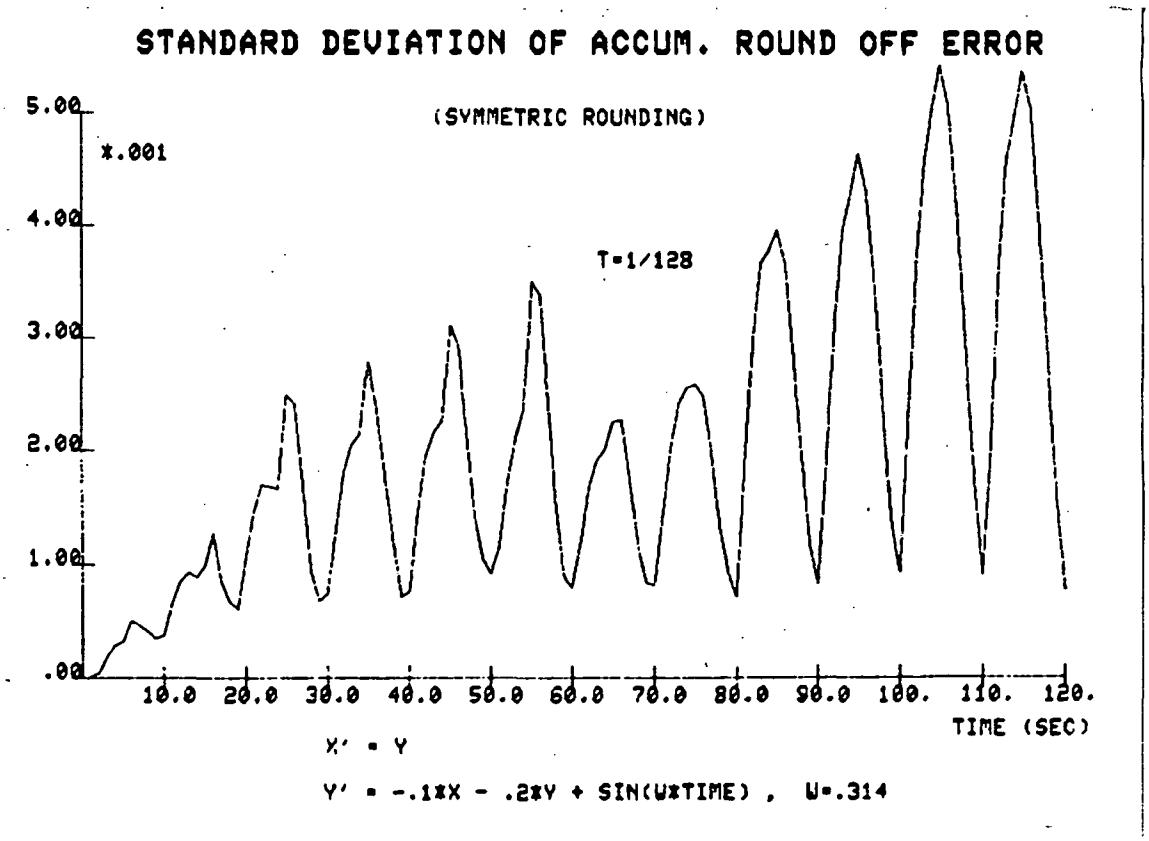

Fig. 3-26. Simulation of a system of second order.

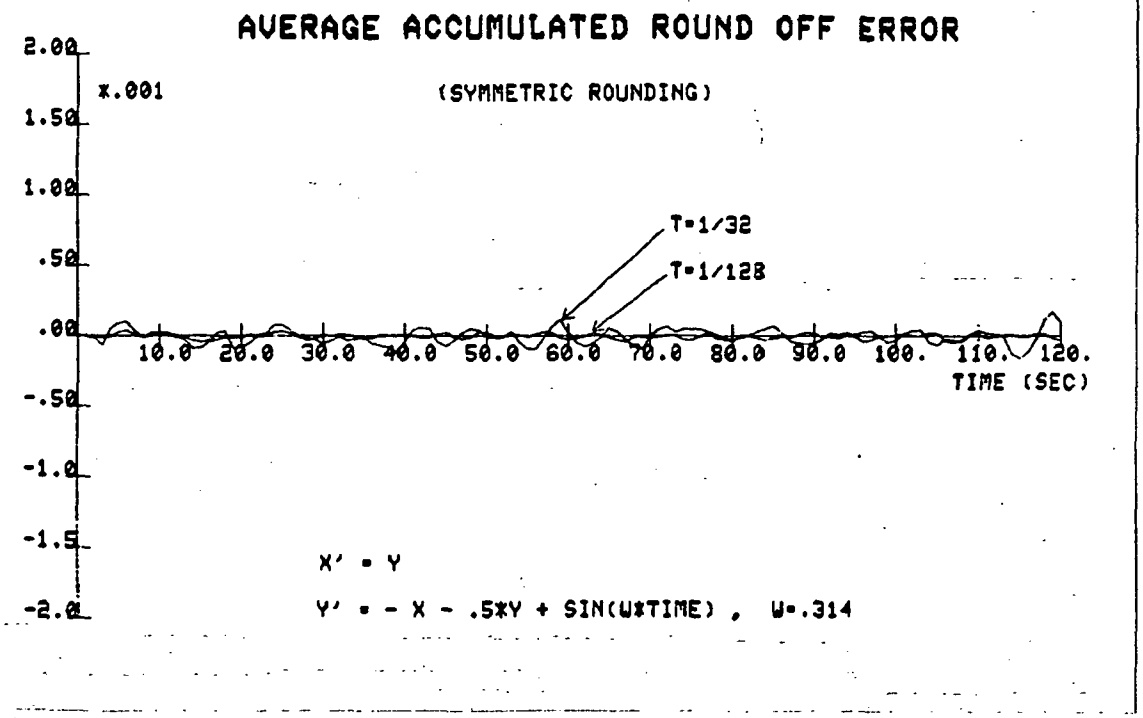

Fig. 3-27. Simulation of a second order system.

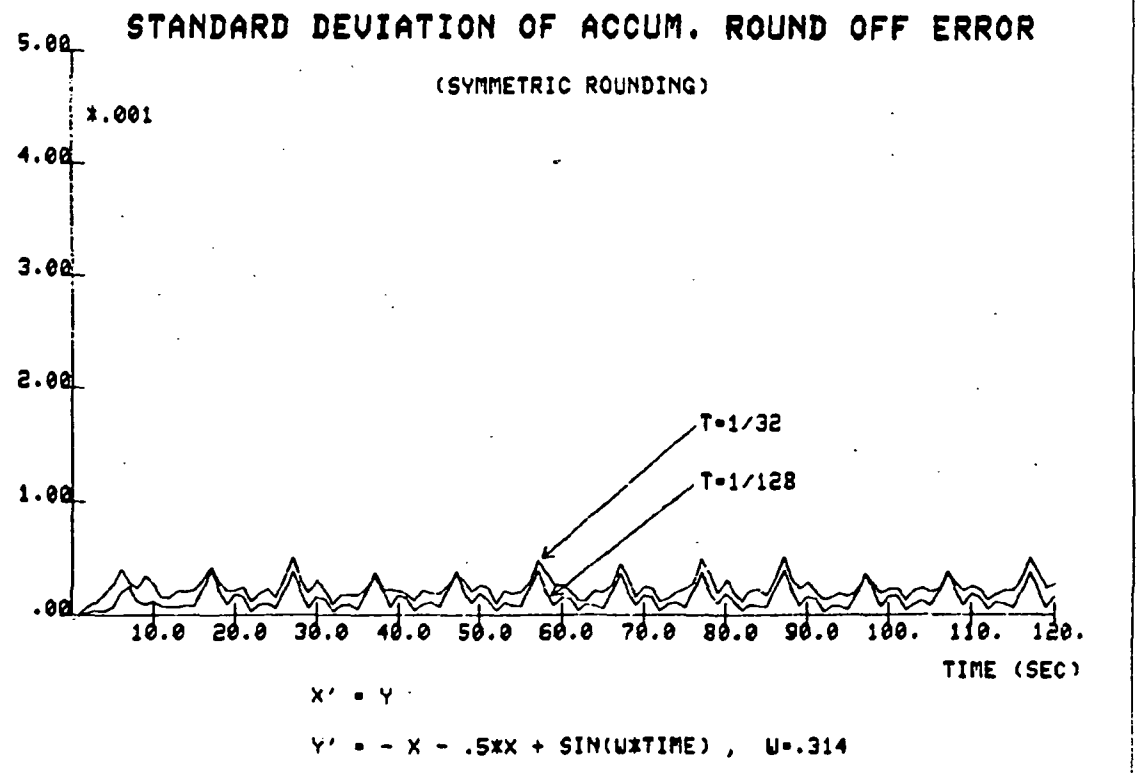

Fig. 3-28. Simulation of a second order system.

3.8 Conclusions

The mathematical approach to the problem of the accumulation of round off errors in numerical integration leads to two general observations. First, all the parameters of the specific simulation under investigation have to be considered; general rules and formulas about the evolution of the accumulated round off error can therefore be established only for a limited range of well defined problems. Second, in many cases this theoretical approach will be hopeless (especially when non-linear equations or large systems of simultaneous equations are involved) because it calls for solving differential equations which will be at least as complex as those defining the original problem; comparative simulation will then be the only way to obtain useful information on this question of round off accumulation.

Regarding the specific question of using short word computers for numerical integration, this research has shown the gain to be realized by a partial use of double precision, or partial double precision in the sense of Henrici. Indeed, if the ratio between the length of the double precision mantissa and the length of the single precision mantissa is large enough, the problem of the accumulation of round off errors will be drastically changed. In the worst case the accumulated round off error at any point of the integration will become independent of the chosen stepsize of integration; in the best case it will slowly decrease with it. This fact certainly opens the door to using microcomputers working with very short sampling periods to perform real time simulation.

At the same time the effects of partial double precision were studied, those of using simple truncation or symmetric rounding were also analyzed. Because of the large difference in magnitude between the expected value of the accumulated round off error and its standard deviation, symmetric rounding (which results in making the expected value go to zero) becomes a necessity with short word computers. It has been shown that this choice combined with partial double precision will result in very small accumulated round off errors (in many instances smaller than the least significant bit) when the coefficient of the dependent variable (L) is not too small compared to the amplitude of the input signal. For this particular analysis the input signal was assumed exactly known (with an infinite accuracy).

A separate look has been given to the question of the accumulation of round off errors when the input signal to the emulated dynamic system is given by an analog to digital converter. It has been found that then a computer with a mantissa slightly longer than the converter word will perform as well as a computer with a much longer mantissa, provided symmetric rounding and partial double precision are used. The consequence of this observation is again that microcomputers will be able to handle simulation as efficiently as much larger and more expensive units when the problem of emulating a particular system requires analog to digital converters.

References

(1) Peter Henrici, "Discrete Variable Methods in Ordinary Differential Equations", John Wiley & Sons, 1962.

(2) J. H. Wilkinson, "Rounding Errors in Algebraic Processes", Prentice-Hall, 1964.

(3) Bede Liu and Toyohisa Kaneko, "Error Analysis of Digital Filters Realized with Floating-Point Arithmetic", Proc. IEEE, Vol. 57, No. 10, October 1969.

(4) Bede Liu, "Effect of Finite Word Length on the Accuracy of Digital Filters -- A Review", IEEE Trans. on Circuit Theory, Vol. CT-18, No. 6, November 1971.

SECTION 4

PREPS ARCHITECTURE AND IMPLEMENTATION

4.1 Introduction

This section is a summary of a one year effort in studying and developing a multimicrocomputer system to be used in parallel simulations. Emphasis was placed on the practicality of reconfigurability, and the compatability of software and hardware. A demonstrative system was built during this year and is described herein.

The system design chosen to meet the objectives is reminiscent of analog computer architecture and uses special computer modules--working in parallel-- to allow a simple definite assignment procedure for various parts of the simulation.

The special representation of the simulated system- which led to the hardware organization- is described first, followed by a complete description of the hardware and software.

This project is based on the premise that a multimicroprocessor simulator has potential for being practical if the required hardware-software combination can be made reconfigurable in a practical manner. A multimicroprocessor system is envisioned as consisting of a large number of interconnected microprocessors and a minicomputer. The rapid technological developments and improvements in LSI microprocessors and arithmetic support chips make a special purpose, reconfigurable, parallel processor dedicated to real time flight simulation an attractive alternative to a standard, large computer complex. It has been

clear for some time that increases in processing speed through technological advances have almost reached an upper limit. Thus, attention has been focused on architectures which exploit parallel processing of data. The advantages are quite significant:

- Optimal (minimal) cycle time reduces the phase shift introduced into the process being simulated.
- Short cycle time decreases truncation error.
- Easy way to expand the system is provided by adding independent computing modules.
- Better reliability achieved by the modular nature of the design, also providing ease of maintenance.

The system to be described suggestes two levels of reconfigurability. Flexible software modules provide the user an initial command language similar to analog computer program diagrams. These modules are designed to simplify the job of generating the simulated system, by providing blocks of programs which occur frequently in a particular form. Simple examples are integrators, weighted sums, non-linear functions, etc. These blocks, although fixed in regular use, can be redefined easily, or others can be added as needed to the system library. Integrations, for example, can be switched from Euler method to Adams-Bashforth method or any other desired one.

Sophisticated arithmetic units connected by a common bus to a master computer comprise the hardware. Reconfigurability is achieved by independent operation of each unit and by the asynchronous mode in which they interchange data. Finally, man/machine programs supply

the relations between software and hardware under specifications given by the user and under optimization algorithms that reconfigure the system to give best results as to speed and performance.

4.2 System Software

A state variable description of the dynamical (possibly non-linear and time varying) system is the basis for the block diagram representation of the simulation system. Independent parts of the differential equations can be evaluated simultaneously, thereby achieving parallelism. For example, consider the simple system given by

$$\dot{X1} = X2 - K4*X1 + K2*X3 \qquad (4\text{-}1)$$
$$\dot{X2} = X3 + K1*R + K4*X1$$
$$\dot{X3} = 2*X2 - K2*X3 + K2*R$$

with initial conditions

$$X1(0) = X10 \qquad (4\text{-}2)$$
$$X2(0) = X20$$
$$X3(0) = X30$$

This system can be represented in block diagram of the kind shown in Figure 4-1. This structure is similar to that suggested for direct execution processors by Korn [3] but is herein used for programming a multimicroprocessor system. This necessitates a "library" which includes multiplications, divisions and integrations as blocks of programs. We can see that the terms K2*X3, K2*R, K4*X1 can be evaluated once (per cycle) although used several times in the system of equations. The block diagram representation eliminates multiple evaluations by its

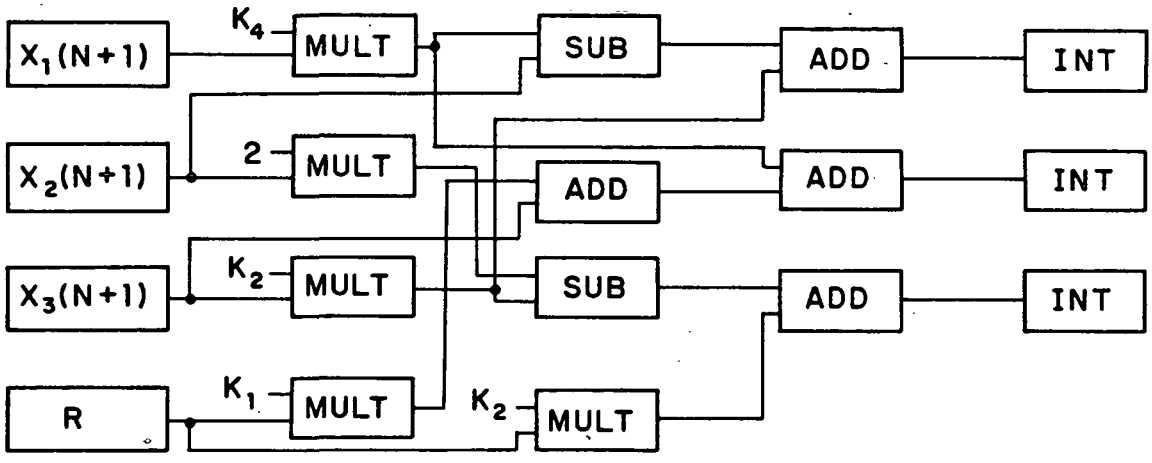

Fig. 4-1. Block Diagram Command Language

nature. The terms K4*X1, K2*X3, K1*R, K2*R can all be in process simultaneously and this is where the multimicroprocessor system has advantages over sequential processing units.

The library blocks are written in the microprocessor's assembly code or preferably microcoded and stored in local PROMs. Assignment of blocks to a processor is done by loading the corresponding command codes into the local RAMs. There they appear as

.

.

MULT

A1

X1

K4

SUB

A2

X2

A1

.

.

.

where MULT and SUB are command codes, A1 and A2 are labels of results and the other are operands. Command codes can be library functions or input-output definitions. I/O codes are carried out by special purpose hardware interfaces hooked to the system bus.

A task list for the example is shown in Table 4-1. This list indicates the fact that tasks may need results from other tasks as their operands.

Table 4-1.

Task List

$A_1 = K_4 * X_1$	$A_8 = A_1 + A_2$
$A_2 = K_1 * R$	$A_9 = A_8 + X_3$
$A_3 = K_2 * X_3$	$A_{10} = A_4 - A_3$
$A_4 = K_2 * R$	$A_{11} = A_5 + A_{10}$
$A_5 = 2 * X_2$	$X_1 = INT\ (A_7)$
$A_6 = A_3 - A_1$	$X_2 = INT\ (A_9)$
$A_7 = A_6 + X_2$	$X_3 = INT\ (A_{11})$

It is noted again that the "library" of tasks was chosen just for the demonstration, and more complicated functions may be programmed by the user. At the preceding step the list is broken down into several parts, and each of the available computers gets some of the tasks to be carried out. A timing diagram of a process (given by the example) is shown in Figure 4-2. The time needed for a completion of a task depends on the hardware speed, and is discussed later. Each library function has an associated execution time which can be measured experimentally when created. Those periods of time are used by an optimization algorithm, which shares the tasks between the computers (before the

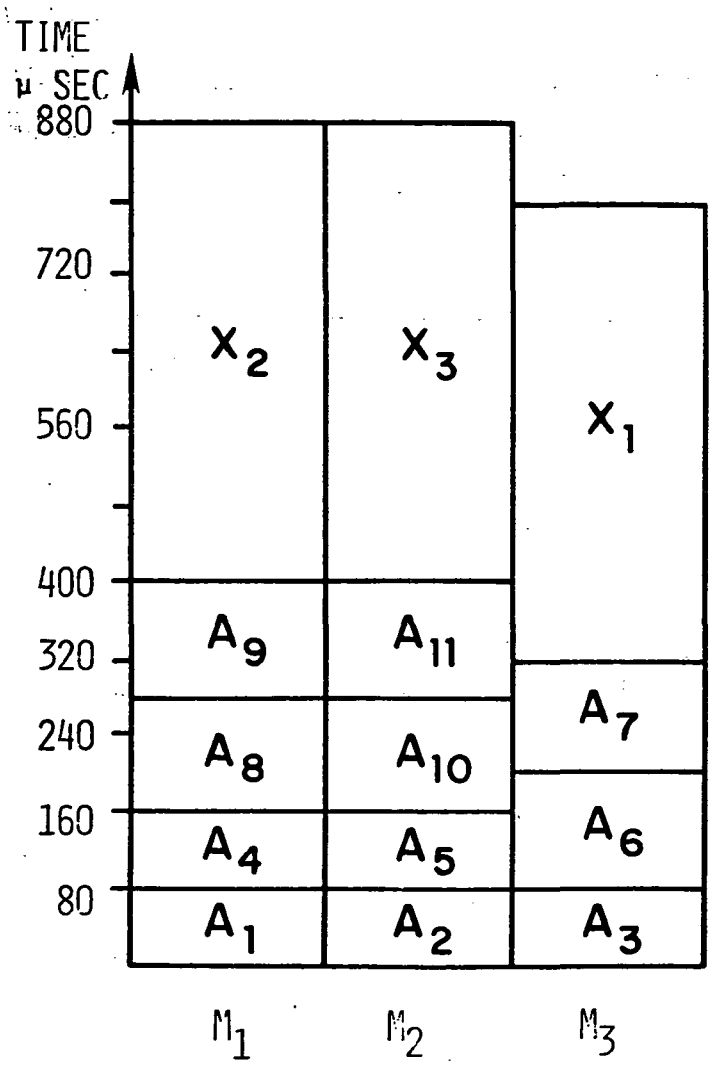

Fig. 4-2. Timing Diagram

beginning of the simulation process), thus minimizing waiting loops which may occur when a task in process needs the result from one that has not been processed yet. In the example shown no such situation occurs, and the simulation program is processed efficiently.

4.3 System Hardware

As viewed from the block diagram representation there are two ways to achieve parallelism:

1. Simultaneous processing of tasks by the available microcomputers.
2. Increasing efficiency in data flow between modules.

The first point indicates that a given simulation system should be analyzed first and broken into subprocesses in which tasks do not depend on each other's results, thus processed simultaneously. The second point implies a special hardware architecutre. If we assume that several tasks may need the same piece of information, an inefficiency may occur when the common bus is occupied more than once to transfer the same information. An interrupt system and preanalysis of the given simulation can solve this problem, based on a "write only" bus architecture. This system is conceptually identical to a shared memory system where all computers use a common memory to read and write. The difference is that instead of one memory module we have an image of it in each local memory. The advantage of the image idea is that although the need for a piece of data arises asynchronously in the computers, which might lead to several transfers of the same data from the memory, the writing occurs only once and the data is distributed in parallel to the computers that need it.

Each of the computers is pre-programmed to detect the appearance of an operand needed by one of its tasks on the common bus. The interrupt system, controlled by the master computer, gives a "bus granted" not to processors that need an input but rather to processors that have intermediate results to output, and are idle until those results are accepted by the other processors. This way several modules can be interrupted at the same time, and perform input operations from the bus. The optimization provides a way to organize the tasks performed by each computer, so that their operands are available in the local memory when needed. The suggested organization that meets the above description is shown in Figure 4-3.

The master computer performs man/machine communications before the simulation begins. It is used as an interrupt controller in the real time run. The number of slave processors is limited by hardware restrictions only.

Two different buses connect the computers to each other:

1. DMA bus.
2. System bus.

The first one is used to load the list of tasks into the local memories during system generation. It also provides access to the internal parts of each module, so the user can check registers, memory bytes and interface status. If the system library is needed to be very flexible, it can be written and held on disc and loaded by DMA into local RAMs for each simualtion.

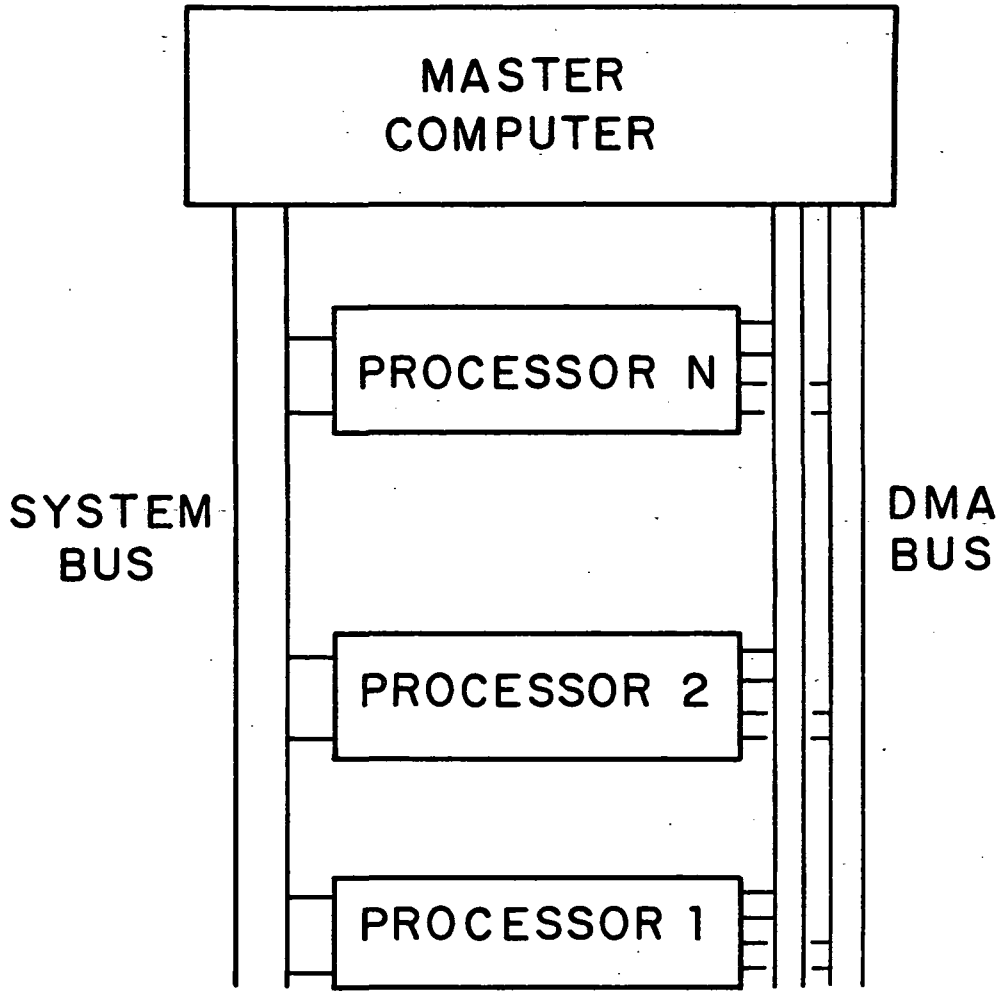

Fig. 4-3. System Organization

The system bus interconnects the computers for transfer of intermediate results, data and status. It includes a 32-bit data path and an 8-bit control path.

Ultimately, the slave computers will be realized with bipolar, micro-programmable bit slice microprocessors because of the associated short cycle times and variable word length. To reduce costs and ease construction of a feasibility experiment, the current version combines the utility of INTEL-8085 microprocessors with the speed of Advanced Micro Devices AMD9511 ALUs. This arithmetic unit is stack oriented and can perform 80-μsec multiplications and divisions, 180-μsec additions and subtractions as well as several trigonometric and logarithmic functions, all done in 32-bit floating point arithmetic (24 bit mantissa, 6 bit exponent, 2 bits for signs).

As shown in Figure 4-4, each computer is comprised of two main parts:

1. Arithmetic logic.
2. Communication logic.

The 8085 microprocessor and AMD9511 provide the number-crunching capabilities. The AMD9511's data path is 8 bits wide, perfectly suited to the internal bus structure and is treated as an interface port. Its activation is done by loading the two operands and the opcode. Its busy line is used to check completion of calculations which, in turn, interrupts the microprocessor. Status bits indicate overflow, underflow, negative argument, division zero by zero and argument too large.

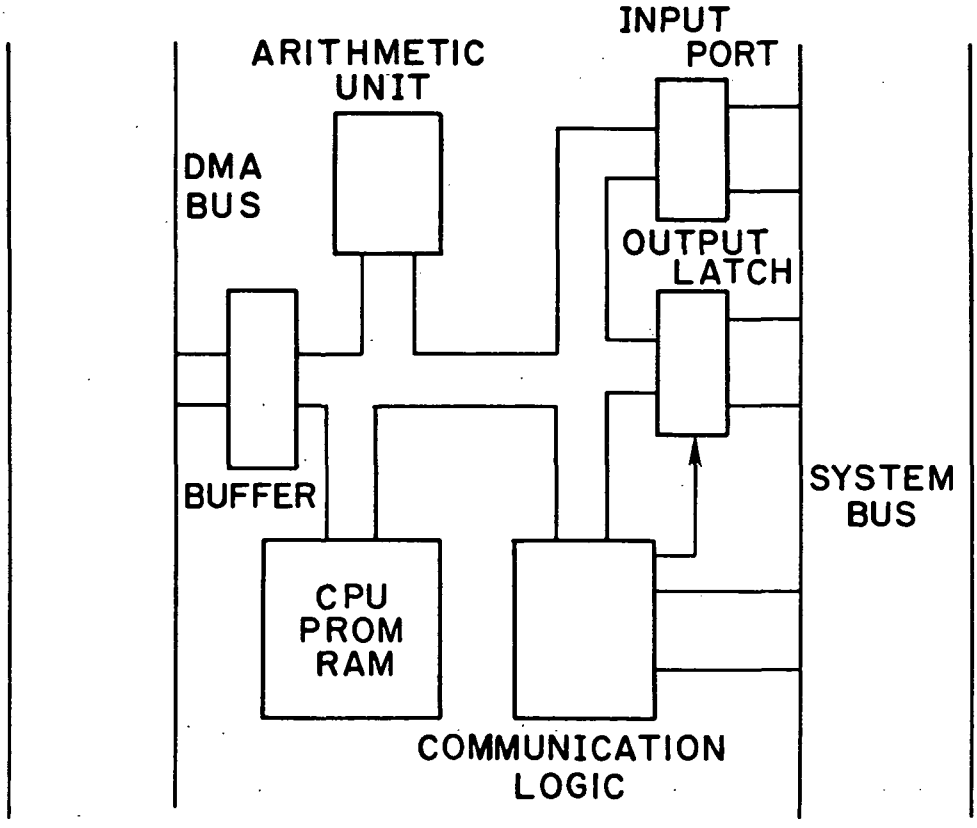

Fig. 4-4. Computing Module

The relatively long time needed by the ALU for calculations is used by the processor to finish the overhead operations like communication with other processors or memory operations.

Upon completion of a task, the results are transferred to the output latches. These are (32 bits wide) tristate flip flops, controlled by the communication logic which, in turn, is connected to the master interrupt controller. The microprocessor goes into a waiting loop if previous results were not taken by the system from the output latches.

The DMA components include gates to the common bus and interrupt logic. A DMA request by the master computer generates an interrupt in the slave processors and a jump to the monitor. This monitor allows major registers and pointers in memory to be examined or changed by the user. The registers are restored and the simulation programs take over upon a start command by the user.

Communication logic handles address and status bits used by the interrupt system. It consists of a one word comparator (6 bits) and programmed lk word comparator. Upon completion of a task, the corresponding task number is loaded by the local CPU into one side of the first comparator. The comparator is activated when the same number appears on the system bus--sent by the master computer which searches for ready tasks. The output gates are opened upon equality, and data flows to the system data lines. Also, a status bit (called s_2) goes low, indicating to the master computer, as well as to the other computers, that data is ready on the bus.

A local operand list created in advance for each computer when tasks are assigned is used to generate interrupts for read cycles. This list consists of task numbers whose results are used by the processor during the simulation cycles. The list is loaded as ones and zeros in the corresponding addresses of a 1 bit wide lk RAM used as comparator. Appearance of one of these addresses on the common bus along with low level status bit (s_2) causes an interruption in the local process and performance of input operation. Input of data by a slave computer is done as follows:

1. Set a status bit (s1) to indicate input operation.
2. Read data (8 bits at a time four times).
3. Reset status.

The data source computer checks s1 reset before releasing the bus, otherwise an input error occurs in the reception. Pre-programming of the lk RAM is done by DMA operation during system generation.

The master computer consists of minicomputer and sophisticated interface to the system bus. This interface is actually a microcomputer which handles the DMA operations and the interrupt control. Four 8 bit data ports are dedicated to communication with the minicomputer, 2 ports are used for DMA transfers, 8 ports are occupied by the system bus and 3 ports are used for general control information. Also, the interface microcomputer functions as a programmable system clock which generates the basic cycle time and counts the true time needed for the multiprocessor system to complete the tasks.

The data base of the interface computer includes an address list of all tasks to be performed. State variables appear first in this list.

The addresses are sent to the system bus during the cycles, providing bus grants, according to the following algorithm:

1. Wait for beginning of cycle signal
2. Reset task counter
3. Ouput next address in list
4. If S2 = 1 go to 3
5. If S1 = 0 go to 5
6. If S1 = 1 go to 6
7. If end of list go to 1
8. Go to 3.

Step 4 is used as feed back to check bus request. Steps 5, 6 indicate the reception of the data by other modules. The algorithm by which the task list is searched can be changed easily from linear to any other desired priority schedule. Finally, the independent nature of the computers may lead to a different concept of implementation: instead of one cycle time for the whole system we can assign a single task to each computer and let it run freely. This demands much larger hardware support, but may reduce discretization error and increase speed significantly. This idea necessitates some changes in the control programs, but the hardware is completely suitable.

Man/machine communicatiosn are performed by the minicomputer. This includes a high level simulation language, an optimization algorithm to assign tasks to available computers and utility programs such as a file manager, DMA driver, etc. A simulation language has not been

developed yet, although the block diagram approach previously mentioned seems attractive. The assignment policy of tasks to different processors has a direct influence on speed and performance. The ideal solution may cause an equal share to be given to each one, and also eliminate idle cycles when one processor waits for results from another processor which has not yeat completed its job. Three algorithms were considered:

1. Critical path determination.
2. Trees.
3. "Fill up" tables.

The first one is based on the fact that each simulation system has a limit of parallelism. This is determined by the longest (most time consuming) path of tasks that feed each other with results, and must therefore be evaluated sequentially. Finding the critical path helps to determine the minimal cycle time. The rest of the tasks must now be examined and shared between the rest of the processors. This adds complexity to the algorithm because it does not provide a way to continue the sharing procedure.

The second one suggests building trees of connected tasks, with weights on the branches, relative to execution times. Again, there is no simple way to determine the relations between the number of available processors and the parts of the trees, especially for tasks whose results appear several times in the calculations.

The third one seems the clearest and simplest to implement, although run time may be longer. The data base consists of the tasks determined earlier. In addition, there is a work list which contains

initial conditions and a library. The library relates experimental execution time to each function. Each processor is represented by a file. A time vector describes the current situation of each processor.

The algorithm assigns the next task to the file whose corresponding element in the time vector is minimal, in order to keep time values in the vector as close to each other as possible. The tasks are chosen from the original list if their operands appear in the work list. A task which is being assigned to a file is also added to the work list indicating that at this stage its calculation is complete. This algorithm does not always determine the optimal task share because of the random way they are chosen from the original list. The timing diagram in Figure 4-2 is a result of using the described algorithm. Each task being processed has its operands ready before the execution, and no need for waiting loops arises.

4.4 Summary

An experimental, reconfigurable multimicrocomputer system for real time simulations has been described. Its features may be further investigated, using the demonstrative system that was built. An effort should be continued to improve software and hardware: high level simulation language must be developed using the block diagram representation and modular I/o interfaces must be designed, based on the existing bus structure and operation.

The current system is constructed of three slave microcomputers and a controller computer connected to HP 2100 minicomputer. Each computer is a modular unit, built on one board and connected to S100

mother board (15 slots). Cost of hardware is less than $2400, including the computers, chassis and power supply.

REFERENCES

1. R. O. Lejune and E. S. McVey, "Computer Word Size Required for Specified Error in Numerical Integration," Proc. 12th Annual Southeastern Symposium on System Theory, May 19-20, 1980, Virginia Beach, Virginia.

2. D. Klien and E. A. Parrish, Jr., "A Reconfigurable Multiprocessor System for Real Time Simulation," Proc. IEEE Southeastcon '80, Opryland Hotel, April 13-16, 1980, Nashville, Tennessee.

3. G. A. Korn, "High-Speed Block-Diagram Languages for Microprocessors and Minicomputers in Instrumentation Control and Simulation," Comput. and Elec. Eng., Vol. 4, 1977, pp. 143-159.

DISTRIBUTION LIST

Copy No.	
1 - 3	National Aeronautics and Space Administration Langley Research Center Hampton, Virginia 23665
4 - 5	National Aeronautics and Space Administration Scientific and Technical Information Facility P. O. Box 8757 Baltimore/Washington International Airport Maryland 21240
6 - 8	E. A. Parrish
9 - 10	E. S. McVey
11 - 12	G. Cook
13	I. A. Fischer
14 - 15	E. H. Pancake Clark Hall
16	RLES Files

222R
0737:ras