プログラミングをめぐって*

——SIP 言語の相互調整；反復法における収束と丸めの誤差——

森口繁一**

1.

今日の記念講演会の目的は，情報処理学会はどういう分野をカバーして，およそどんなような事を取り扱うのかというようなことについて，いろんな例を出すことだと承知しております．それで私は今までの二つの大変おもしろい有意義な講演とはまたゆき方をちょっと異にしまして，現在あるいは少なくともごく近い将来の電子計算機を使っていく上に必要なことについて一，二の例をお話したいと思います．

まず第一は計算機のメーカーさん達が次から次へと新しい種類をお作りになる．それは高橋さんが最初にかかげられたような目標を持っていろいろと前進されるわけですが，その際にプログラマーにとってはなはだ困ることには，機械の言語がその都度変わって出てくる．それをいちいちまた覚え直さなくてはいけない．翻訳機ですとさっと新しい単語と文法を読み込めばすむのかも知れませんが，人間の頭というのは内部にそういうものを半固定記憶みたいにして作りあげねばならないものですから，大変そのプロセスに時間がかかるのみならず，一度非常に確かに覚えてしまったものは将来の学習に際しては邪魔になる事さえある．それが，機械が進歩するのが早ければ早いほどプログラマーにとっての悩みの種になると思われるのであります．現にそういう悩みを悩んでいる人達も少なくないのです．

そこで私共，ちょうど一年ぐらい前に相談致しまして，これは少なくとも日本では一つの教育用の共通の言語を作ろうではないか，計算機の言語として共通のものを作って，基本教育をいわば統一的にやりましょう．それから個々の機械に移るに当って，なるべく滑らかにつながっていくようにしましょう．そのためにはその教育用の共通の言語のほかにそれぞれの機械について専門の言語を作らなければならない．それが教育用の言語と同じ文法に従って，語彙の点もなるべく共通の部分を多くして学習しやすいようにしたい．そして混乱がおこらないようにしたい．そういうことを相談したのであります．

一年の間にかなりその方向に向って仕事を進めましたし，これからやろうとしているものも二，三現われておりますので，その現状をこの機会にお話しまして，皆さん方の御批判を得，望むらくは御支持を得て，その仕事を今後も強力に進めていくようにしていきたいと思っている次第であります．

2.

ここにビラを用意致しましたが，[redacted]に書いてありますのが SIP 100 という教育用の一般言語で表わしたプログラムの例であります（**第1図**）．この辺がごく簡単な加減乗除の例でありまして，XA/200. 200 番地にある数を破算した Acc. に入れなさい，A/201. その上へ 201 番地の数を加えなさい，その上へ 202 番地の数を加えなさい，T/203. でその結果を 203 番地へしまいなさいといったような調子でできているのであります．またこの 201 とか 202 とかいう具体的な番地のかわりに，A, B, C, というような記号を使うことも許すことになっております．

XA/200.	*a*	XA/A.	*a*
A/201.	*b*	A/B.	*b*
A/202.	*c*	A/C.	*c*
T/203.	*s*	T/S.	*s*
XA/200.	*a*	XA/A	*a*
B/201.	*b*	B/B.	*b*
T/204.	*d*	T/D.	*d*
LMD/200.	*a*	LMD/A.	*a*
XMA/201.	*b*	XMA/B.	*b*
T/205.	*p*	T/P.	*p*
XA/200.	*a*	XA/A.	*a*
LMD/201.	*b*	LMD/B.	*b*
DIV/~.		DIV/~.	
T/206.	*q*	T/Q.	*q*
HJ/PPO.	停　止	HJ/PPO.	停　止

第 1 図

* Some topics related to programming——coordination of SIP languages: convergence and rounding error in iterative processes, by Shigeichi Moriguchi (University of Tokyo).

** 東京大学工学部教授

こういうふうに演算部にも番地部にも記号が使えるという意味で symbolic という名前で呼んでおりまして，Symbolic Input Program というのが SIP という言葉のおこりであります．そういう記号で書いていれます．そしてその記号をなるべく共通な部分を多くしておけば学習にも転換にも容易であろうと考えたのであります．

もうちょっと込み入ったものになりますと，たとえば積の和をつくりたい，20個の数の積の和をつくりたいといったような時には，**第2図**のように

```
I1S/19.          "19"→Ind 1        I1S/19.
→ PP1)                             PP1)
LMD/200+I1.      ai                LMD/A+I1.    ai
MA/300+I1.       bi                MA/B+I1.     bi
—I1N/PP1.        (Ind 1)≠0ならば飛ぶ  I1N/PP1.
```

第 2 図

とやっておきまして，プログラム・ポイント1番をここに定義して，LMD/200+I1. MA/300+I1. とやりますと，200番地以降に入っている数字と300番地以降に入っている数字とが掛け合わされて次々と加算されていく．I1N/PP1. というのでここへもどる．もどるたび1ずつ下っていきますが，そういうふうにしてそれが20回繰り返される．その間に20個の積の和ができあがるといったような表題になっています．

いろんな機械について，こういう機能は大抵あるわけですが，それが違った言語でなるべく呼ばれないように，そう書けばどの機械でもだいたいいくようにしたいというのが主旨であります．もちろん，この200というかわりにAと書いたり，300というかわりにBと書いたりすることも許しています．こんなものがだいたい今 SIP 100 として教育用に使っている言語の命令の形であると御承知願いたいのです．

その次ですが（**第3図**），四則演算につきまして現在教育用 SIP 100 という言語で使っている演算コードがこのような形をしておりますが，それに対して NEAC-2203 専用の SIP 101 で使っているのがこれで，HITAC-301 専用の SIP 102 という言語で使っているのがこれです．互いに共通なものが多いことがおわかりでしょう．

それから今製作中のもので HIPAC 101 B という機械のための HISIP 101 B というのがございますが，これもこれらと共通のものが多いことがわかります．それから TOSBAC-3121 というのも東芝で現在製作中のものですが，そのための専用 SIP を，これは仮に SIP-T と呼んでいるわけですが，その中でもこれらと似た機能がある．それは SIP 100 と同じものを採用する予定になっております．

割算というのが種々さまざまでございまして，機械によって，非常に約束が違う．この場合に約束が違うものが同じ言葉で書かれているということは間違いのもとだと思いますので，そういうものはむしろ言語が違った方がよいのであります．しかし，それがなるべくは類推でいけるようにしていきたいと考えているわけでして，この部分でこの相互の関係というものの一例をみていただきたいのですが，SIP 100 では，DIV/～．こういう表現に致しまして，割られる数は必

四則	(共通) SIP 100	(NEAC-2203) SIP 101	(HITAC-301) SIP 102	(HIPAC-101B) HISIP 101 B	(TOSBAC-3121) SIP-T
加減	XA/*n*.	XA/*n*.	XA/*n*.	XA/*n*.	
	A/*n*.	A/*n*.	A/*n*.	A/*n*.	A/*n*.
	XB/*n*.	XB/*n*.	XB/*n*.	XB/*n*.	
	B/*n*	B/n.	B/*n*.	B/*n*.	B/*n*.
絶対値加減	XAŌ/*n*.	XAŌ/*n*.	XAŌ/*n*.	XAŌ/*n*.	
	AŌ/*n*.	AŌ/*n*.	AŌ/*n*.	AŌ/*n*.	
	XBŌ/*n*.	XBŌ/*n*.	XBŌ/*n*.	XBŌ/*n*.	
	BŌ/*n*.	BŌ/*n*.	BO/*n*.	BŌ/*n*.	
乗算	XMA/*n*.	XMA/*n*.	XMA/*n*.	XMA/*n*.	XMA/*n*.
	MA/*n*.	MA/*n*.	MA/*n*.	MA/*n*.	
	XMB/*n*.	XMB/*n*.	XMB/*n*.	XMB/*n*.	
	MB/*n*.	MB/*n*.	MB/*n*.	MB/*n*.	
除算	DIV/～.	DBY/*n*.	XAD/*n*.	XAD/*n*.	XAD/*n*.
			AD/*n*.	AD/*n*.	
			XBD/*n*.	XBD/*n*.	
			BD/*n*.	BD/*n*.	

第 3 図

	SIP 100	SIP 101	SIP 102	HISIP 101 B	SIP-T
load	LMD/*n*.	LMD/*n*.	LMD/*n*.	LMD/*n*.	LMD/*n*.
store	T/*n*. XT/*n*.	T/*n*. XT/*n*. TMQ/*n*.	T/*n*. XT/*n*.	T/*n*. XT/*n*.	T/*n*. TL/*n*.
round	R/∼.	R/∼.	R/*n*.	R/*n*.	
shift	SL/*n*. SR/*n*.	SL/*n*. SR/*n*. SC/*n*.	SL/*n*. SR/*n*. SLA/*n*.	SL/*n*. SR/*n*. SC/*n*.	SLU/*n*. SRA/*n*.
no. op.		NE/∼.	NE/∼.	NE/∼.	NE/∼.
jump	HJ/*n*. J/*n*. JP/*n*. JM/*n*. JZ/*n*.	HJ/*n*. J/*n*. JP/*n*. JM/*n*. JZ/n. JC/*n*. JE/*n*. JŌ/*n*.	HJ/*n*. J/*n*. JP/*n*. JM/*n*. JZ/*n*. JAŌ/*n*.	HJ/*n*. J/*n*. JP/*n*. JM/*n*. JZ/*n*. JŌ/*n*.	HJ/*n*. J/*n*. JLU/*n*. JHU/*n*. JE/*n*.

第 4 図

ず Acc. に入っていて，割る数は MD レジスタに先に入れてあるという約束になっております．

それに対して，日電の 2203 のための専用 SIP では，DBY/*n*. というふうに表わすことにしております．これは *n* という番地は割る数（Divisor）の所在を示すわけですから DBY/…(divide by…) とやっておけば，それで割るということがよくわかるというわけです．これはその番地部が割る数であるということを示唆する言葉として採用してあります．それに対して SIP 102 とか HISIP 101 B とかいう日立系の機械ではルールが違いまして，まず Acc. にある数に番地部によって指定された数を加えてから割算をやる．割る数の方はあらかじめ MD レジスタに入れてあるというルールに従っておりますので，たとえば加えてから割るというのは Add and Divide, AD と書きます．

それから Acc. をはらってから加えてそれを割るという場合には，XAD というふうに書く．そうすると，XA や A がわかっていますとその後へ D をつければそういう演算をやった後で割算をすればいいというふうにこれは延長として理解することができるのであります．そしてしかもこのコードの特長は相当よく反映していると思います．

こんなふうに同じ機能は同じ言葉で表わされ，違った機能は適当に違った記号で表わされるということをわれわれは調整と呼んでいるわけです．相互に調整された言語の系統を持ちたいということは，こんなふうに進行しておるのです．

なお，いくつかの明瞭な例を少しお目にかけますが（**第 4 図**），LMD/*n*. というのは MD レジスタに *n* 番地にある数を入れる，のっける（load する）という命令ですが，これは全部に共通．Store *n* 番地（T/*n*.），これは Acc. の内容を *n* 番地へしまう，その前に Clear（X）がついた場合が XT です．それから SIP 101 では TMQ/*n*. という MQ の内容を store するという命令があるのですが，それは TMQ と書いておけば類推でわかるだろう．それから TL/*n*. も store lower という命令があります．Round もだいたい似ております．それから SL（Shift Left），SR（Shift Right）も共通ですが，この東芝の機械は SLU というふうに，Upper Acc. だけを shift left するということをこういう形で使っています．

SC(Shift and Count) というのは SIP 101 と HI SIP 101 B が共通でございます．SLA（Shift Left Around）とか SRA（Shift Right Around）というようなものもだいたい似たような形であります．NE（No Effect）は SIP 101，SIP 102，HISIP 101 B，SIP-T，全部に共通であります．HJ(Halt Jump) とか J(Jump)，JP(Jump Plus)，JM(Jump Minus)，JZ(Jump Zero)，というものも共通であります．

それから JE(Jump Equal) というのがありますが，MD レジスタと Upper Acc. とが等しい場合に Jump

	SIP 100	SIP 101	SIP 102	HISIP 101 B	SIP-T
Index	I*i*S/*n*. I*i*J/*n*.	 ISJ/I*i*/*n*.	I*i*S/*n*.	I*i*S/*n*. I*i*J/*n*.	I*i*S/±*m*+I*i*.
	I*i*Z/*n*. I*i*N/*n*.	JIZ/I*i*/*n*. JIN/I*i*/*n*.	J*i*P/*n*. J*i*M/*n*.	I*i*Z/*n*. I*i*N/*n*.	I*i*E/I*j*/*n*. I*i*N/I*j*/*n*.
	TI*i*/*n*. LI*i*/*n*.	TI/I*i*/*n*. LI/I*i*/*n*.		TI*i*/*n*.	TI*i*/*n*. LI*i*/*n*.
入出力装置指定	SEL/□.	SEL/□.	SEL/□.	SEL/□.	
入力	XRN/*n*. XRH/*n*.	XRN/*n*. XRH/*n*. RH/*n*.	XRN/*n*. RN/*n*.	XRN/*n*. XRH/*n*. RH/*n*.	RI/*n*.
出力	ON/*n*. OS/□. OH/*n*.	ON/*n*. OS/*n*. ŌH/*n*.	ŌN/*n*. ŌS/□. ŌH/*n*.	ŌN/*n*. ŌS/□. ŌW/*n*.	PŌ/*n*.
抽出	XEA/*n*. EA/*n*.	XEA/*n*. EA/*n*.	XEA/*n*. EA/*n*.	XEA/*n*. AND/*n*.	BA/*n*. BM/*n*.
番地演算		XAA/*n*. AA/*n*. XBA/*n*. BA/*n*.	XAA/*n*. AA/*n*. XBA/*n*. BA/*n*.	XAA/*n*. AA/*n*. XBA/*n*. BA/*n*.	

第 5 図

をやるというわけです．JHU (Jump High Upper) という形は Upper Acc. の内容が MD レジスタの内容よりも higher であれば Jump がおこる．JLU (Jump Low Upper), lower であれば Jump がおこるというような違いであります．これくらいにしておけば共通な部分と違っている部分とがちょうどよいくらいになっておるのであると考えられます．

Index register 関係もだいたい似ております (**第5図**)．東芝の機械では Ii と Ij とを直接比較致しまして，等しい場合には番地へ飛ぶといった機能がありますが，それは I*i*E/I*j*/*n*. の形に書いておけばよいのではないかと思います．Not Equal の場合には I*i*N/I*j*/*n*. TIi (Store Index i), LIi (Load Index i) などというのもだいたい今までのものと共通です．入出力の指定には SEL (Select) というものを使います．XRN (Clear Read Numeric) とか，XRH (Clear Read Character) も共通です．

だいたいこの辺までの命令はおわかりと思いますが，こういうものを読んでいくときに，終りまで読んでしまわないと最初の文字を何と読むかわからないというようなのは初め覚えるときに大変骨が折れる．これは恐らく心理学上裏付けもできるだろうと思いますが，従来の Symbolic Code の中にはそういうものが非常に多い．特に三文字に揃えるために無理をしたような機械では，それが非常に多いように思います．わ

	SIP 100	SIP 101	SIP 102	HISIP 101 B	SIP-T
独特のもの		LMQ/*n*. IAU/I*i*/∼. IBU/I*i*/∼. ITA/I*i*/∼. SLH/*n*. SRH/*n*. LMH/*n*. XAH/*n*. AH/*n*. TLU/*n*. BP/∼. LT*i*/*n*. TT*i*/*n*. FŌN/∼. FŌF/∼. NŌR/∼.	XA/MD. A/MD. XB/MD. B/MD. XAU/*n*. AU/*n*. XBU/*n*. BU/*n*. JQŌ/*n*. XEB/*n*. EB/*n*.	SLD/*n*. SRD/*n*. BHJ/*n*. JSW/*n*. JPD/*n*. JMD/*n*. CŌN/□.	LU/*n*. LL/*n*. TMD/*n*. XU/∼. XL/∼. XMD/∼. JNE/*n*. I*i*H/I*j*/*n*. I*i*L/I*j*/*n*. XI*i*/∼.

第 6 図

れわれはそういうことはしませんで，頭から大抵は順ぐりに一つずつ読んでいけば読み方がはっきりしているというふうにしております．修飾は後につけます．頭にXがくれば必ず Clear であります．Index 関係は大抵 I がつき，頭に T がくればこれは大抵 Store であり，L がくれば Load であるといったようにしてあります．

したがって番地演算で Address にこの *n* という数を加えるといったようなものも，XA, A, XB, B, という四つの基本型に A (address) という修飾字を後へつけるということで表わしているわけです．XEA

(Clear Extract Add), および EA (Extract Add), これは SIP 100 と同じものが全部共通に使われているわけですけれどもたとえば HISIP 101 B では AND という演算がありまして Acc. の現在の内容と n 番地の内容を，これは2進法の機械ですからビットごとにすべての桁を比較しまして AND という演算をやった結果を Acc. に入れるものです．これは今までに例がございませんので，AND という新しいコードを考えます．

それから東芝の機械は，これは十進法の機械ですけれども，Bitwise Addition, Bitwise Multiplication という機能がございますので，そういうものは BA. BM, という記号で表わしています．

以上のようにだいたい類推でわかるものが多いという点を御注意願いたい．SLH とか SRH とかいう最後にHの来たのはみな Character（文字演算）であるということに注意すれば，あとは類推でわかります．この程度で一つの機械から他の機械へ移るとき，この SIP 仲間ですと転換は非常にやさしいのではないかと考えております．

3.

なお，番地部の表わし方は，絶対番地でたとえば 1200 とかいうふうに書きます（**第7図**）．それから先ほどのようにAからZまでの記号を使います．またはA+3というように，Aという番地から三つ先という表現もできる．それから PP 1（Program Point 1 番）というようなものがあります．これが1番から

番	XXXX		
	A+XX		
地	PPXX+XX		
部	＿＿＿＿ +I*i*		
制	SIP 100.	PP 1	CANCEL/□.
御	L_0 ( )	A	CHECK.
指	A ( )	WS.	STOP/CHECK.
令	START ( )	HALT.	
数	±. 1 2 3 4 5.——小数		
値	± 1 2 3 4 5.——整数		

第 7 図

99番まで許されます．その Program Point 3 番から三つ先というようなときには PP 3+3 と書いてよろしい．以上のどれにでも後に +I i というのがつきますと Indexi のそのときの内容を加えたものが実行番地になるというようになります．

この辺はほんの少しの時間でみなマスターすることができます．こういう表現はすべての SIP に共通に許されるということになっておりまして，覚えなおす必要はないようです．

それから制御指令，まず頭書の SIP 100 ですが，このたぐいのものを必ず一つの新しいプログラムの冒頭につけて，それがたとえば SIP 100 の言葉で書かれたプログラムであるということを機械に示すことにしてあります．機械の方はそれを受け入れるプログラムが入っておれば無事に通しますが，さもないと言葉が違うといって拒否して print out して止まってしまうようになっております．こういうようなことでチェックをします．それと同時に今までの記号やプログラム・ポイントの定義を全て解消して，新しいプログラムを受け入れる準備を致します．現在これだけを読んで実行するのに数秒かかっております．

格納開始番地は L_0 (　　)というように書いて，ここへ，たとえば200と書けば，200 番地以降に以下の命令文が入るというような指令になります．A（300）と書けばAという記号は300番地と定義したことになります．START（100）と書くと100番地の命令から実行を開始せよという指令になります．それから PP 1）というものを途中へ挿入致しますと最初の例にも出てきましたが，ここの所へ PP 1 を定義したことになります．それから記号番地Aの定義も，途中へ A）を挿入することによって随所に定義することができます．作業用番地を1個保留したいときには WS. を脇に添える．読み込みの途中でテープを一時止めて，restart ボタンを押せば SIP の読み込みを再開するようにするには HALT. としておけばよいのです．

このような制御指令はいかなる言語を使っても必ずこういう一揃は必要であるわけです．それは現在までの SIP はできているものも，これから作ろうというものも全部これは共通になっております．その点は大変具合がよくなっています．

それから CANCEL/□. は今までに定義した A ならAという記号番地の値をいったん Cancel して，次に新しい定義で使おうというときにこういうものを入れます．CANCEL/A. と書いておくと A の定義は解消されます．といいますのは，SIP では A の定義が出ないうちに前の方の命令で番地部にAというのを使っておってもかまわないようにできておりまして，これは非常に高級な操作を機械がやってくれるようになっているのですが，こんなことがありますので今まで

のAとは定義をかえるというときにはCancel指令をはっきり入れませんと混乱がおこるわけです.

今までの続きとしてうめていってよいのか,あるいは次に新しく定義されるまで待っていなければいけないのかということを機械が判別する必要があります.そんなわけで消すときにはCANCEL/□.という制御指令を入れます.

それから,CHECK.というのを勝手なところに挿入しておきますと,そこのところで,チェック・サブルーチンへのつなぎが植えつけられることになりまして,計算が進行してここまできますと,そのときのAcc.の内容がprint outされるというふうになっています.それは実際この種の言語でつくられたものを検査(debug)する,プログラムをテストするというような場合に大変役にたちます.CHECK.と書いておくだけでチェックのためのやや複雑な操作が自動的に行われます.

チェックがすっかりすんで,もう大丈夫という状態になったら今度はSTŌP/CHECK.というのを読ませますと,CHECK.という指令が全部無視されるということになりまして,この点も教育用のSIPでは大変喜ばれているようでございます.

4.

以上が第一の話でありまして,SIP言語の相互調整という部分であります.相互調整の主旨と現状とを話したわけですね.そこでいわば情報処理学会と結びつけていいますと,やはりこういう仕事は大勢の方々の協力によって全国的に一本になった形でおしすすめないとうまくいかないのですね.誰かが自分の研究室でコツコツとやっていればよいというような仕事ではないと思います.そういうわけですから,こういう全国的な学会が,こういうものの調整の主体になっておしすすめていくのがよいのではないでしょうか.ですから将来たとえばSIPに類似の言語をつくりだそうというようなときには,そのセンターと連絡をとっていただいて,今までのものと同様によく調整された言語として開発していただくとよいのではないかと考えている次第であります.

それから一般用の教育用の共通言語としてのSIP 100は,現在電子協の教育には全面的に採用されていまして,これで基礎コースをやっているわけですが,相当うまくいっておるように聞いております.ですから将来ももしそれがよろしいのならば,方々でおやりになる講習会や,あるいは大学の講義とかいうようなものも,できるならばSIP 100を使ってやるようにしていただくとお互いに便利ではないかと思っているわけです.

もっともまたやっているうちに言語の単語とか文法とかに関して,もうちょっと直した方がよいではないかという意見も生まれてくるかも知れませんが,そういうものも,まあたとえば情報処理学会の専門分科会の一つにそういうことを扱う部門があって,いろんな苦情や,改善意見などを常時受け付けて審議するといったような体制ができるとさぞよかろうと考えているわけです.

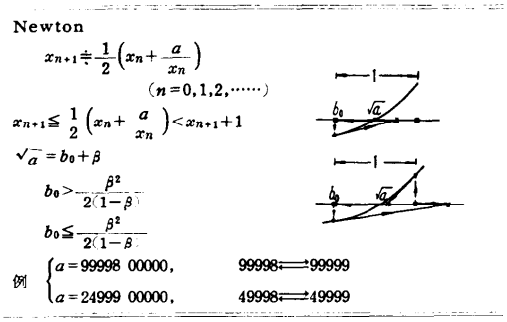
5.

SIPに関する部分はこれくらいに致しまして,次は反復法における収束と丸めの誤差という話に入ります.実は,理事会で今日の講演の相談をしましたときに,私がこのSIPの話を持ち出しましたら,それも結構だが,その他に数値計算に関係したことも少しは入れてもらいたいというような御希望が出たというので,この話を用意したわけです.

これはプログラマーの悩みといいましても,また別の面でありまして,実際に計算をプログラムして進行していくときに,丸めの誤差が思わぬ障害をきたすことがあります.特に反復法で方程式の解などを求めていきますときに,最後にこれで収束したという判定をどんな形で行なうか,という部分に丸めの誤差がきいてきて,ほんとならループを脱出してくれると思われるのに,意外にもループから出てくれない,といったようなことがおこる.それがまた必ず出てくれるように,誤差の限界を大幅にとって計算すると,みすみす精度を損をするというような事態がときどきおこります.それを防ぐためには,丸めの誤差が収束にどんな影響を与えるかというようなことを,あまり忙しくない人がゆっくりと体系的に考えて,その考えた結果をときどきはこういう学会で講演をしたり,雑誌にのせたりするのがいいんじゃないかと思います.そこでこういうようなものも学会では受け付けるという例として,ちょっとここに名のりをあげてみたわけです.

6.

電子計算機で平方根を求めるには,Newtonの反復法を用いるのがふつうです.それはこういう公式によるわけです(第8図).$\frac{1}{2}(x_n+a/x_n)$を計算した結果



第 8 図

が，x_{n+1} という整数と $x_{n+1}+1$ との間にこういう形でおさまっている場合に，この整数値 x_{n+1} が次の近似になるという形に書けるわけです．最低位の1を1としてここへ表わしているわけですが，そこでこれをよく解析してみますと，このような反復法の最後の状態というのは二とおりあるということがわかります．

この辺は，やはりかなり数式を運転しないと出てこないのですが，この $\sqrt{a}$ が求めたいものですが．それが b_0 という整数部分（一番下を単位としての整数）プラス端数 β からできているとしまして，その b_0 と β との間にこういう関係がありますと，それは一つの値に収束致します．そうするとそれは b_0 そのものですから，これは一番具合のよい値にほんとうに収束することになる．このときは最後の状態はこういうループを描くことになりまして，b_0 でのこの点を求めまして，ここでの接線をひいて横軸との交点を求めます．これがこの部分です．それがこの点と次の点の間にありますから，結局これに切り捨てを行うと，ここへくるというわけで，同じ b_0 が何度も相次いで現われます．こんなときには Equal Jump で脱出することができるわけです．

この平方根の場合はこれが圧倒的に多いです．大抵の場合これになりますから収束するのがふつうですが，しかし収束しない場合があります．しない場合があるということは，経験的にはなかなかわからないと思います．たとえば，すでに公刊されている大変評判のよい本で，Equal Jump で脱出するようなプログラムが，模範プログラムとして出ているのがございますが，あれは恐らく a としてこういう値を選んだらだめになるだろうと思っております．その a は5桁の機械ですと .24999 00000 というような a をとるのです．そうすると，その平方根は .49998 と .49999 の間を往復致します．

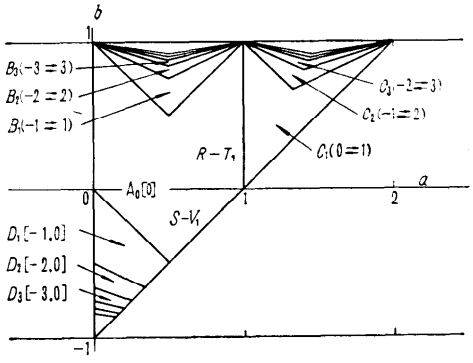
それはなぜおこるのかといいますと，b_0 のところで接線をひくとここへきます．それを切り捨てを行いますとこの点になります．ここで接線をひくとここへきて，切り捨てを行うとここへきます．これを繰り返すわけですね．

こういうことが，詳細にこれを考えて，つまりこういう部分に顕微鏡をあてたような調子で数式をあやつると出てくるわけです．私はこれを大変興味深いと思いまして，一昨年の数学会で発表致しまして，この種のことを含む一般理論ができたらさぞ愉快だろうといいましたら，広島の古部さんから，実はそれに関係した話を以前に論文に書いたことがある，というようなお話があり，あとで別刷を送っていただきました．一般論として大変おもしろいものが古部さんの研究として出ております．

その後，去年あたりきたアメリカの雑誌の中に von Neumann, etc. の論文があって，その中にこの平方根のものなどが非常に詳しく出ておりました．やっぱりこの種のものを注目している方もかなりおられるということがわかって大変心強い次第であります．

7.

ところで，そのときに発表したもう一つの例がこれ（**第9図**）でありますが，これは方程式としては $x_{n+1} = a - bx_n$ という単純な一次方程式，こういう反復をやらせたときに最後の状態がどうなるかということを吟味したわけです．それは方程式を反復法で解くとき



第 9 図

の一つの一番単純なモデルだと思って，これを詳しく調べてみました．そうしましたら a を横軸にとって b を縦軸にとった場合に，こんなきれいな絵が出てまいりました．もちろん最低位の単位を1として，a の方

は表わしてあるわけですが，b の方はほんとうの値です．b が -1 から 1 までの間にありますと，あのプロセスがふつうの意味で収束します．

つまり丸めを無視した場合に，あれが収束するための必要十分条件は b の絶対値が 1 より小さいということでございますが，そういう場合に a の値を標準化しますと，この三角形の中までは簡単に追い込むことができます．そして，その三角形の中がこういうふうにぬりわけられた，いろんな部分にわかれておりまして，たとえば，この矢羽根みたいな格好をした領域は 0 と書いてありますように，いかなる値から出発しても最後の状態は必ずこの 0 というところへきて収束します．だから Equal Jump で脱出できるわけであります．それに対してこういう矢の頭のような状態の中では，最後の状態は -1 と 1 との間を往復するというようになって，これは Equal Jump では脱出できません．

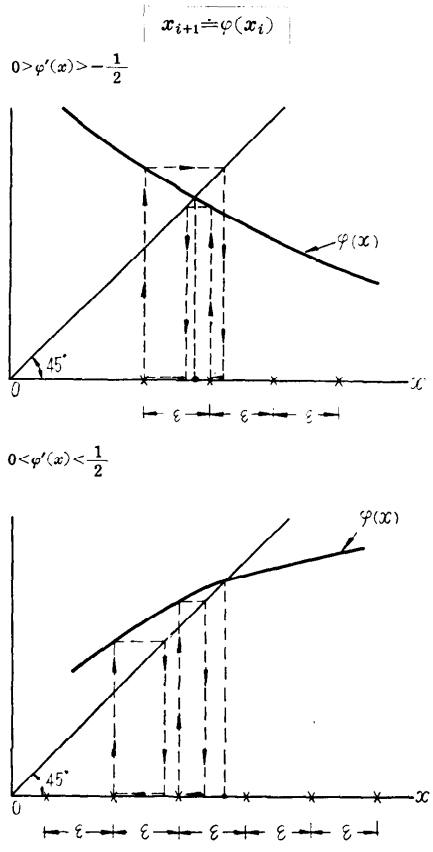
その上は誤差を最小単位の 1 とおいておいたのでは脱出できない状態であります．最小単位の 2 だけの幅で振動がおこりますから，どうしても 2 だけは余裕をもって収束を判定しないと脱出できないわけです．その上は -2 と 2 の間を往復するから幅が 4 になる．その上は -3 と 3，こういうところへきますと -1 と 2 との間を，-2 と 3 との間を往復します．

この部分が 0 と 1 との間の往復であります．それから下の方へきますと，今度はこういうひしゃげた菱形の部分は -1 という値に収束するか，または 0 という値に収束します．その最後は必ず一定の値になるのですけれども，その一定になる値が二とおりあります．左の方からくれば 0 になるし，右の方からくれば -1 になる．そういう状態にあります．この部分は 0 と -2 という二つの値のどちらかになる．といったような部分が，ずっとわかれてまいります．

ここでおもしろいのは，b が $1/2$ 以下であって $-1/2$ よりも上でありますと，つまり b の絶対値が $1/2$ 以下でありますと，これはここのところで一定の値にならないにしてもその振動の幅は 1 であります．振動がおこったにしても，その振動の幅は 1 しかないということがいえます．ですからもし b の絶対値が $1/2$ を越えないということであれば，最後は 1 だけの幅を許しさえすれば，ループから脱出することができるといえます．これは精度をあまり犠牲にしないで要領よく脱出する方法だろうと思います．

負の場合でも，$-1/2$ と 0 との間に b がありますと必ず収束して，その値は二とおりありますけれども，その幅は最小単位の 1 を越えませんから，まずまず精度の点で問題はないというようなものであります．それが $1/2$ を越えたり，$-1/2$ よりももっとマイナスの値になったりしますと，だいぶ考えなければならないことがおこってきます．というようなことがこれで詳細にわかるわけです．

こんなことを詳細にやります意味は，これ自身が本当に実用に役立つからというつもりではありませんで，こういうものをよくやっておけば，もうちょっとややこしい実用的な場合に見とおしがきくだろうと考えてやったことであります．そこでその後もうちょっと実用的なものを考えたものがこれ（**第 10 図**）でありまして，$x_{i+1}=\varphi(x_i)$ という形で反復をやるとして，この $\varphi'(x)$ が，考えている領域で 0 と $-1/2$ の間にあるということが保証できたならば，先ほどここでいった，振動がおこるにしても幅は 1 であるという結論



第 10 図

が，やはりあてはまるのです．その点はこれを解析的に証明しておきさえすれば，ちょっとした拡張でこの一般の場合に拡張することができます．そしてこの結果ここに書いたような振動がおこり得ます．

こことここの間がεと書いてあります最小単位でありまして，これより小さい端数は取り扱わないということになっているわけですが，この点とこの点の間を往復することは，このような場合におこり得るけれども，これより大きな幅の振動はこの条件のもとではおこらないということが保証されるとか，こういう条件がなり立つならば，最後にはこの値にくるか，この値にくるかのどちらかであって，それ以外の値にはいきません．結局最小単位だけ離れた二つの収束値があって，そのどちらかへ必ずいくといったようなことが一般にいえます．

そこでこういうことの応用例としましては，これ（**第11図**）は前に水晶の振動に関して一度やったことがあるのですが，こんなような反復をやった場合に，cは必ずより小さいコンスタントであって K/C が 1/2 より小さければ，先ほどの条件がなり立つということがいえます．実際これは0.1くらいの大きさでありまして十分収束します．これは先ほどの条件がありますので，幅1の許容範囲で判定して脱出することができるという場合に属します．こういう一般的な保証があればプログラムを組むときに大変気が楽だろうと考えているわけです．

例 1	$x_{n+1}=\pi m-\arctan(c\tan k x_n) \qquad c<1$ $\lvert\varphi'(x)\rvert \leqq \frac{k}{c}<\frac{1}{2}$
例 2	$\frac{dy}{dx}=f(x,y)$ $y_{n+1}=y_n+\frac{h}{2}(f_n+f_{n+1})$ $\lvert\varphi'(y)\rvert=\left\lvert\frac{h}{2}\frac{\partial f}{\partial y}\right\rvert<\frac{1}{2}$
例 3	$\frac{dy}{dx}=f(x,y)$ $y_{n+1}=y_{n-1}+\frac{h}{3}(f_{n-1}+4f_n+f_{n+1})$ $\lvert\varphi'(y)\rvert=\left\lvert\frac{h}{3}\frac{\partial f}{\partial y}\right\rvert<\frac{1}{2}$

第 11 図

それから例2，例3というのは微分方程式の数値解法の場合でありますが，例2は梯形則による反復法でありまして，こんな場合はこういうものが1/2より小さければ幅1以内に収まるという条件がなり立つわけです．これもふつう，微分方程式の解法の場合にはなり立っている条件だと思います．これがおかされるようでは多分hが大き過ぎるのですね．それから例3はMilne のⅦとよばれるシンプソンの 1，4，1 を使った反復法の公式ですが，この場合には対応するものがこんなものになります．

いずれにしてもそういう条件さえあれば反復の収束は幅1で判定してよろしいという一般理論があるというような話になっているわけです．もちろんこれはほんの局部的な結果ですが，こんなような話題も情報処理学会の扱う範囲に含まれるという一例としてお話致しました．

参考文献

1) 森口・清水・松谷：“プログラミング教育用 SIP 100 システム”，日本数学会秋季大会応用数学分科会講演予稿（その二），1959年10月．
2) 森口繁一：“SIP 100 による基本プログラミング”電子工業振興協会，第2版，1960年7月．
3) Sigeiti Moriguti: “Notes on the numerical convergence of iterative proceeses”, Contribution to Probability and Statistics (Olkin et al. eds.), Stanford University Press, 1960, p. 309-321.