MMS 26 - 1
(1976. 11. 16)

会話形連続系システム・シミュレータ

臼井 支朗　　浅見 秀雄　　池谷 和夫
（名古屋大学工学部）

はじめに　連続系（continuous system）のシミュレーションにアナログ計算機が使われることはよく知られているとおりである．アナログ計算機は，システムを表現するブロック・ダイアグラムと計算機の結線との対応が直観的にわかりやすく，解が高速に得られるという利点を持つ．

一方，近年になって汎用ディジタル計算機がめざましく発展し，それを用いたディジタル・シミュレーションが盛んになって来た．ディジタル・シミュレーションには，アナログ計算機に比較して，精度が高い，スケーリングの必要がほとんどない，複雑な非線形要素も容易に実現できる，装置をこわす心配がないなどの利点がある．数値積分には四次 Runge-Kutta-Gill 法など高精度な方式が考案され，CSMP，GASP などのシミュレーション言語も数多く開発されている．

しかし，これまで開発されているディジタル・シミュレータは，シミュレーション言語を覚えなければならないものが多く，会話形式で人間が直接介入しながらシステムの記述，変更を試行錯誤的にくり返すことができるというアナログ計算機の長所を生かしたものはあまり多くはない．

我々は，名古屋大学大型計算機センターの第2システム（富士通 FACOM230-35電子計算機，F6232Aグラフィック・ディスプレイ装置）を用い，オンライン会話形式で結果がただちに得られ，システムの記述が簡単なディジタル・シミュレータ“NUSSII”（Nagoya University Systems SimulatorII）を開発した．

NUSSII概要　NUSSIIでは，表1に示すとおり，30種類のブロック（計算要素）が用意されている．ブロックは最大3つの入力端と1つの出力端を持ち（入力端のないもの，出力端のないものもある），最大4個のパラメータを指定できる．パラメータとは，各ブロックごとに必要に応じて指定するデータであり，数値型（実数型）のものと文字型（4文字）のものがある．

これらブロックと接続線を用い，GD（グラフィック・ディスプレイ）画面にブロック・ダイアグラムを描くことができ（図1-a），計算開始指令と共にGD画

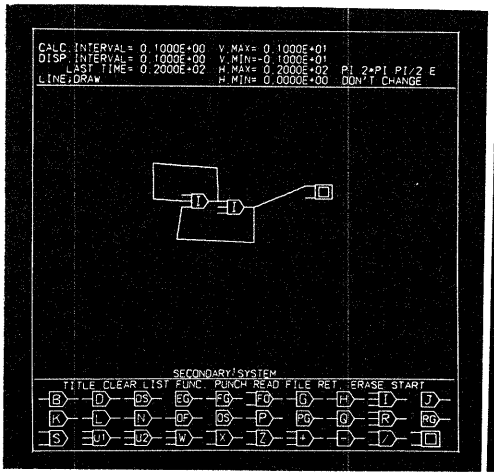


図1-a　DIAGRAM PHASE

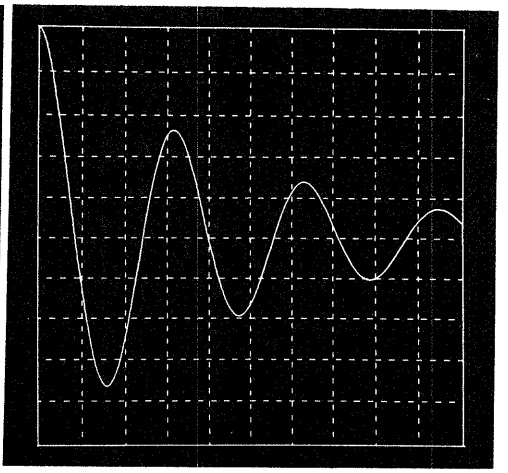
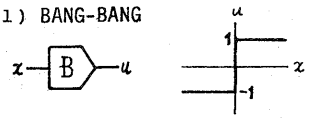


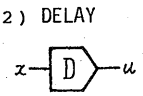
図1-b　ANALYSIS PHASE

表 1　NUSS II ブロック　(注：tは時刻，P_1,P_2,P_3,P_4はパラメータ)

(1) BANG-BANG

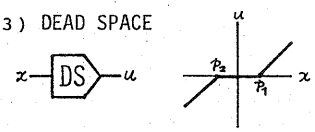


(2) DELAY

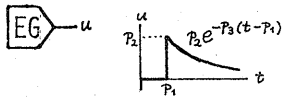


$t<P_1$; $u=0$
$t\geq P_1$; $u=x|_{t-P_1}$

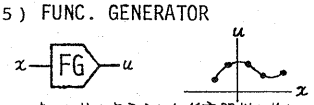
(3) DEAD SPACE



(4) EXP. GENERATOR

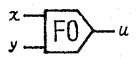


(5) FUNC. GENERATOR



表形式で与えられた任意関数に基き三次補間によって出力を計算する.

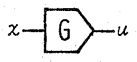
(6) FUNC OPERATOR



$u=|x|, \sqrt{x}, \sin x, \cos x, \tan^{-1}x, e^x, x^y, \log_e x, \log_{10} x, \tanh x$

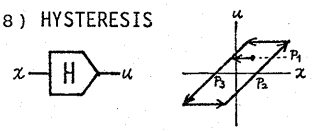
関数の選択はP_1(文字型)により行う.

(7) GAIN



$u=P_1 x$

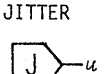
(8) HYSTERESIS



(9) INTEGRATOR

$u=P_1+\int_0^t (x+P_2y+P_3z)\,dt$

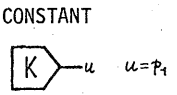
(10) JITTER



区間[P_3,P_2]の一様乱数
平均値P_2 標準偏差P_3の
正規乱数

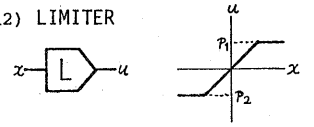
乱数の選択はP_1(文字型)により行う.

(11) CONSTANT

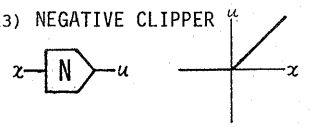


$u=P_1$

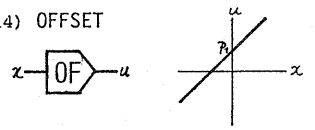
(12) LIMITER



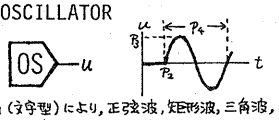
(13) NEGATIVE CLIPPER



(14) OFFSET

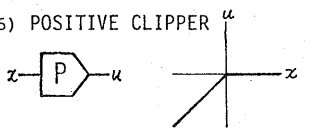


(15) OSCILLATOR

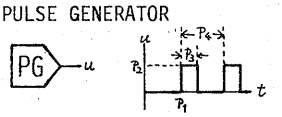


P_1(文字型)により，正弦波，矩形波，三角波，のこぎり波の選択が可能.

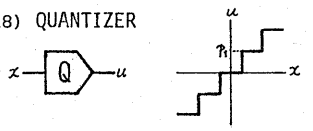
(16) POSITIVE CLIPPER



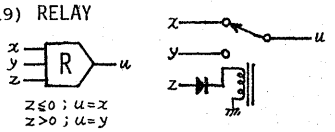
(17) PULSE GENERATOR



(18) QUANTIZER

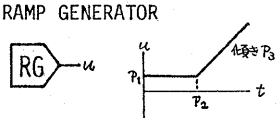


(19) RELAY

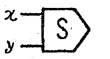


$z\leq 0$; $u=x$
$z>0$; $u=y$

(20) RAMP GENERATOR



(21) STOPPER

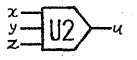


x>yになったら
計算を停止する.

(22) USER'S BLOCK (1)

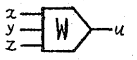
使用者がサブルーチンを
作って機能を定義する.

(23) USER'S BLOCK (2)



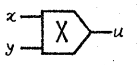
使用者がサブルーチンを
作って機能を定義する.

(24) WEIGHTED SUMMER



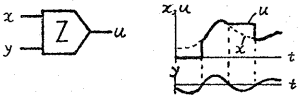
$u=P_1x+P_2y+P_3z$

(25) MULTIPLIER

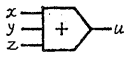


$u=xy$

(26) ZERO ORDER HOLD

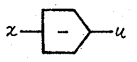


(27) SUMMER



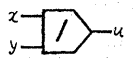
$u=x+y+z$

(28) SIGN INVERTER



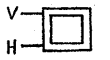
$u=-x$

(29) DIVIDER



$u=x/y$

(30) DISPLAY SCOPE

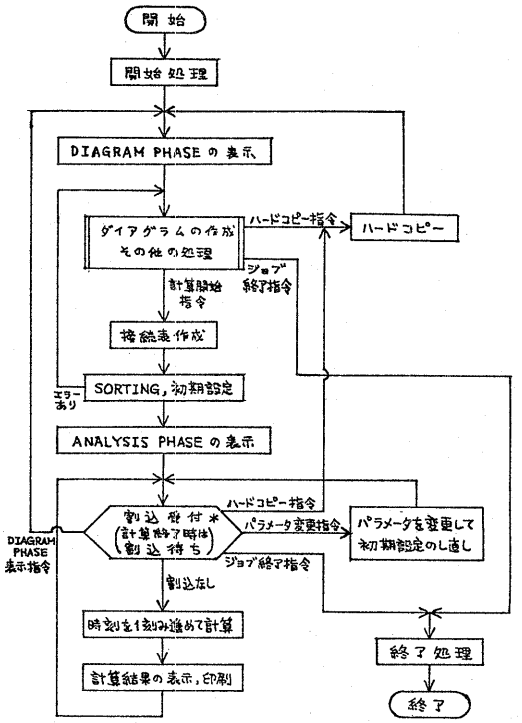


CRTに表示すべき解析結果を読み込む.
Vは垂直入力．Hは水平入力で，Hに接続がなければ横軸は時間軸となる.

面からダイアグラムは消え，代りに計算結果がグラフの形で時々刻々表示される（図1-b）．また，ダイアグラムの表示に戻る指令により，再びダイアグラムが画面に表示される．GD画面にダイアグラムを記述する段階をDIAGRAM PHASE，記述されたシステムを解析し，結果を表示する段階をANALYSIS PHASE と呼ぶ．

DIAGRAM PHASEで行える操作は次のとおりである．

- ダイアグラムの作成，変更，ならびに表題のキーイン．
- 計算条件（計算時間間隔，表示時間間隔，最終時刻），スケール（ANALYSIS PHASEの画面上端，下端，右端，左端の値），各ブロックのパラメータの指定．これらは画面上段に表示されている．
- ダイアグラムの表形式によるライン・プリンタ出力．
- FUNCTION GENERATOR（任意関数発生器）用データ・カードの読込み，
- ダイアグラムのカード出力およびカード入力．
- ダイアグラムの一時記憶および再呼出し（10枚まで）．
- ダイアグラムのXYプロッタへのハードコピー．
- 計算開始指令．



＊ 割込受付または割込待ちとは，ライトペンまたはファンクション・キーによる割込の識別記号を読み出すこと．「受付」ならば割込がなくても次へ進むが，「待ち」ならば割込が発生するまで待つ．
ここでいう計算終了時とは，計算が正常に終った場合のほか，エラー発生時，STOPPER起動時を含む．

図2 NUSSII フローチャート

ANALYSIS PHASEで行える操作は次のとおりである．

- 計算結果のライン・プリンタ出力．
- 計算結果の実線表示／点線表示の切替え．
- 計算の一時停止，再開．
- 画面の鎖線グリッドの表示，消去（消去時には四辺に目盛が表示される）．
- グラフのXYプロッタへのハードコピー．
- パラメータ変更指令（あらかじめ指定されたとおりにパラメータを変更し，時刻0から計算をし直す．結果は重ねて表示される）．
- DIAGRAM PHASEへの復帰指令．

図2にNUSSIIのフローチャートを示す．

ダイアグラムの作成　使用したGD装置は，画面上の何も光っていない任意の位置をライトペンで指定できる機能は有していない．その代り，トラッキング機能（十字形のトラッキング・シンボルがライトペンに追随する）により，画面上の任意の位置を指定できる．ダイアグラムの作成にはこの機能を用いている．

ブロックの表示・登録は，画面下段に並んだブロックのサンプルの端子をライトペンでピックアップすることによって行う．そのときのトラッキング・シンボルの位置に，ピックアップされた端子の先が来るように，ピックアップしたのと

同じ種類のブロックが表示・登録され，自動的にブロック番号が割り当てられる．ただし1か所に2個以上のブロックが重ねて表示されることはない．ブロックは80個まで表示・登録できる．

線をひくには，トラッキング・シンボルを引っぱって止める（ライトペン・スイッチをOFFにする）か，端子またはすでにひかれている線をピックアップする．このとき，トラッキング・シンボルの初めの位置から止まった位置まで，あるいは端子または線の先端（線の場合はピックアップした場所に近い方の先端）まで線がひかれ，自動的に線番号が割り当てられる．なお，線をひくかひかないかの指定を切り替えることができ（どちらの指定になっているかは画面に表示されている），ひかないという指定になっていれば，トラッキング・シンボルを動かしても線はひかれず，また端子や線をピックアップしてもその先端にトラッキング・シンボルが出るだけである．線は160本までひくことができる．

なお，線を「接続する」には必ず端子または線をピックアップしなければならない．なぜなら，トラッキング・シンボルを端子や線の先まで引っぱって来て線をひいたとき，たとえつながったように見えても，座標が1（ラスタ・ユニット——約0.3mm）でも違っていればつながったことにはならないからである．

また，消去指定をすることにより，ピックアップしたブロックや線を消去することができる．

パラメータを指定するには，まずブロックの本体をピックアップすると，画面上部の計算条件とスケール指定の表示が消え，代りにパラメータの名前と値が表示されるので，指定したいパラメータをピックアップし，キーボードから入力する．

<u>ダイアグラムのメモリ内部での表現</u>　ブロックについての情報は，メモリ内部ではブロック番号を添字とする配列に記憶される．これをブロック表と呼び，図3に示すように，ブロックの種類番号（種類を表わす数字），出力端座標，入力先，パラメータ，出力値，および補助配列から成る．入力先とは，各入力端が何番のブロックの出力端につながっているかを示すものであり，そこに記された番号のブロックの出力値を読み出すことにより，入力値が得られる．この部分はダイアグラムの接続に関する情報であり，特に接続表と呼ぶ．接続表は，各ブロックの出力端座標と接続線の先端座標に基いて自動的に作成される．なお，使われていない番号の所は，種類番号に0が入れてある．

接続線については，線番号を添字とする配列に両端の座標を記憶する．これを接続線表と呼ぶ．使われていない番号の所は，x_1に32767が入れてある．

ブロック表

ブロック番号 (subscript)	種類 (integer)	出力端座標 (integer)		入力先 (integer)			パラメータ (real)				出力値 (real)	補助用 (real)
		x	y	1	2	3	1	2	3	4		
1	9	-204	51	0	1	2	0.0	-0.2	-1.0	*	0.0	*
2	9	57	48	1	0	0	1.0	0.0	0.0	*	1.0	*
3	30	311	60	2	*	0	*	*	*	*	*	*
4	0	*	*	*	*	*	*	*	*	*	*	*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
80	0	*	*	*	*	*	*	*	*	*	*	*

接続表

接続線表

線番号 (subscript)	先端座標 (integer)			
	x_1	x_2	y_1	y_2
1	-204	-33	51	63
2	57	221	48	75
3	-294	-366	51	59
4	-366	-363	59	131
5	-363	-216	131	132
6	-216	-204	132	51
7	-294	-290	-286	-361
8	-290	49	-361	-363
9	49	57	-363	48
10	32767	*	*	*
⋮	⋮	⋮	⋮	⋮
160	32767	*	*	*

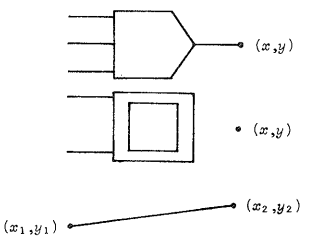


図3　ブロック表と接続線表

接続表の作成　入力先を探すには，入力端から線をたどって出力端のつながりを見つければよい．パターン認識機能の優れた人間の目には何でもないことのように思えるが，計算機にこれをやらせるには巧妙な工夫を要した．そのアルゴリズムの基本を次のように考えた．

いま着目している入力端に直接または間接に（他の線を介して）つながっているすべての線の集合を，その入力端を根元とするtreeとみなす．そのtreeには，どこか1か所だけに出力端がつながっているか，さもなければどこにもつながっていない．2か所以上につながっていることは，出力端どうしの接続があることになり許されないが，そのような接続はないことを前提とする．

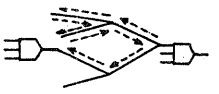


図4　接続線のたどり方の例

さて，出力端のつながりを探すには，図4に示す例のように，根元から順に枝分れをたどり，枝の先まで行きついたら後戻りして別の枝分れを探すという手順をくり返す．そのアルゴリズムのフローチャートは図5に示すとおりである．

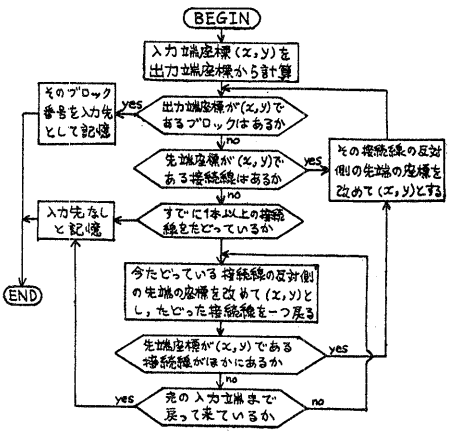


図5　入力先を探すアルゴリズムのフローチャート

このフローチャートについて説明する．まず，入力端座標はそのブロックの出力端座標から簡単に求められる．次に，「出力端座標が(X,Y)であるブロックはあるか」の判定には，使われているブロックすべてについてブロック番号1から順にチェックしていく．このチェックにひっかかったブロックの番号が入力先として記憶されるが，実際には出力端を持たないブロック（STOPPERおよびDISPLAY SCOPE）であったならばもちろん除外する．「先端座標が(X,Y)である接続線はあるか」の判定には，使われているすべての線の両端について線番号1から順にチェックしていく．yesと判定されれば，チェックにひっかかった線の反対側の先端の座標を改めて(X,Y)とする．これでtreeの分岐が1段終った．このとき，今第何階（根元から何本目）の枝をたどっているかを整変数に記憶する．そしてその整変数を添字とする整配列にたどった線の番号を記憶するが，第2先端が根元に近いならば負符号をつけておく．

「すでに1本以上の接続線をたどっているか」の判定でnoならば，入力端には出力端も接続線もつながっていなかったことになるが，yesならば，枝の先まで行きついてかつそこには出力端がつながっていなったことになる．そこで再び今たどっている線の反対側の座標を(X,Y)とする．これで枝を1段戻ったことになる．そして「先端座標が(X,Y)である接続線がほかにあるか」という判定によって，座標(X,Y)の点からの別の枝分れを探す．もしあれば同じようにたどっていくが，なければ「元の入力端まで戻って来ているか」という判定に移る．yesならば枝分れをすべて調べ尽しても出力端がなかったことになるが，noならばさらに戻る．このとき，前述のように線をたどった順にその番号，ならびに符号によって向きが記憶されているので，その番号の線の根元に近い方の先端の座標を(X,Y)とし，同じように別の枝分れを探していく．

なお，線をたどるたびに，線番号を添字とする論理配列に論理値TRUEを記憶し，すでにたどった線はチェックにひっかからないようになっている．したがって，いまn番の線につながっている線を探すとき，当然n番自身も座標が一致してい

るが，チェックにはひっかからない．また，ループ状の接続があってもそれにはまりこむこともない．

この接続表作成アルゴリズムでは，「出力端座標が(X,Y)であるブロックはあるか」というひとつの判定のために，すべてのブロックについてIF文でチェックするという原始的な方法を用いており，多大なステップ数を要している．計算機に連想記憶のような機能を持たせればステップ数は大幅に減少できるはずであるが，今のところ人間の視覚に忠実な方法はほかにないように考えられる．しかし，座標をすべて固定小数点で扱っているため，IF文実行時の減算は速く，NUSSⅡで使用可能な最大ブロック数80個（入力端数240），最大接続線数160本を用いて試験的に作ったダイアグラムで約6秒，実用的には，それほど複雑でなければ1秒以下で作成できる．接続表作成に要する時間は，ブロックの入力端数や接続のしかたなどによって大幅に異なってくるので，単にブロック数や接続線数だけで所要時間を計算できるものではない．

計算開始指令をしてからANALYSIS PHASEに切り替るまでの時間を短縮するためには，DIAGRAM PHASEにおけるCPUの空き時間を利用して接続表を作るという方法が考えられるが，いつ計算開始指令が来てもよいように何度も接続表を作り直すために，CPUのむだな占有時間が増すことになり，得策でないと思われる．

本アルゴリズムの問題点は，出力端どうしの接続という誤りがあってもそれを検出できないことである．というのは，treeをたどって最初に出力端のつながりを見つけたらすぐに入力先決定としてしまうからである．すべての枝分れをたどって，2つ以上の出力端がつながっていないことを確認すればよいのであるが，枝分れをたどるには前述のごとく多大なステップ数を要するので，さらに時間がかかることになる．むしろそのようなチェックは，パターン認識機能の優れた人間にまかせた方が得策であろう．

なお，一度接続表を作成したら，ダイアグラムの接続の変更や全面的な書替えがない限り，それ以後の計算開始指令では改めて作成することはせず，所要時間の短縮をはかっている．

各ブロックの計算順序　NUSSⅡでは，数値積分には四次Runge-Kutta-Gill法を用いている．この方式は単に時間刻みΔtをかけるだけのEuler法に比べ，1刻みあたり4倍の手間がかかるが，粗い刻み幅でも高精度を保つことができる．ところが，刻み幅が粗いと，各ブロックの計算順序がでたらめであれば順序しだいで結果が変ってくるという矛盾を生ずる．そこで，ある規則によって計算順序を定める必要がある．これを自動的に行う機能をsorting機能という．ここでは，簡単なEuler法を念頭に置きながらsorting機能の説明をするが，この考え方は四次Runge-Kutta-Gill法でもそのまま通用するものである．

ある時刻$t_k=k\Delta t$ $(k=0,1,2,\cdots)$における状態（各ブロックの出力）を計算するとき，積分器の入力値として読み出される値はt_{k-1}における状態でなければならない（$k=0$のときは例外）．したがって，最初に積分器，次にそれ以外のブロックについて計算する．しかし，積分器の直後にまた積分器がある場合，初めに前段についてt_kにおける出力を計算してしまうと，後段の入力としてはt_kにおける状態が読み出されることになり，計算のルールに反する．そこで，積分器について計算した出力値はまず補助配列に入れておき，出力値用配列にはt_{k-1}における値を残しておく．そして，すべての積分器について計算が終ってから，補助配列から出力値用配列に移す．次に積分器以外のブロックについて計算する．入力端を

持たないブロックについてはただちに初期出力を決定できるが，入力端があるときは，すべての入力端の入力先の初期値が決定されていることを確認してから初期出力を決定する．入力先の初期値が未決定ならばあとまわしにする．ただし入力先がないならば，入力値は常に0であるとして初期入力は決定済みの扱いとする．こうしてすべてのブロックについて初期値が決定されるまでこの手順をくり返す．なお，初期値決定済みの目印として，ブロック番号を添字とする論理配列を用い，初期設定の終ったブロックに対応する配列要素に論理値TRUEを記憶する．また，計算の順番を添字とする整配列に，初期設定完了したブロックの番号を記憶しておき，あとは各時刻ごとにその順序でブロックの出力を計算していく．

ところで，積分器以外のブロックのみでループ接続を成していると，全初期状態の決定まで至ることはできない．すなわちsorting不可能となる．アナログ計算機ではこのような接続をしてもさしつかえないが，NUSSⅡでは許されない．このときは，sorting不可能というエラー・メッセージを表示する．

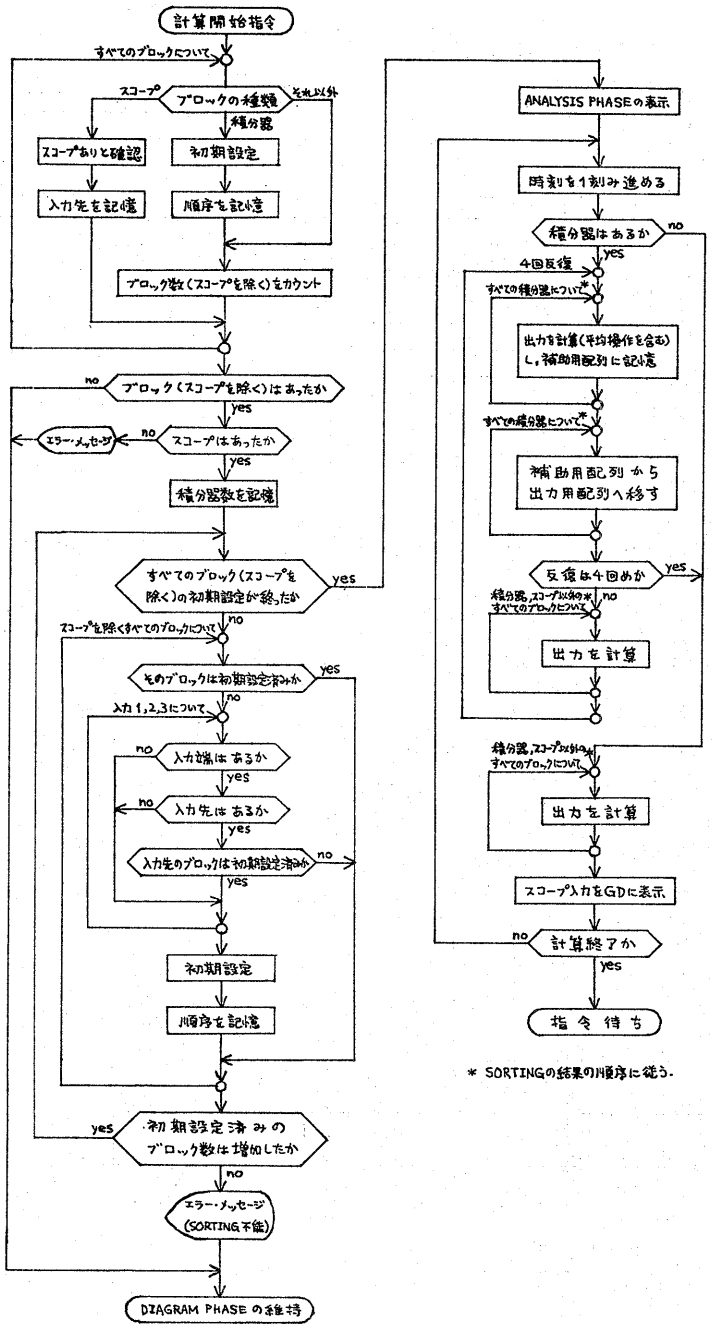


図6　Sorting，初期設定，解析のアルゴリズムのフローチャート

図6に，sorting，初期設定から解析に至るアルゴリズムのフローチャートを示す．

セグメント構造　NUSSⅡはBOSⅡ(Batch Operating System)というオペレーティング・システムのもとで作動するが，BOSⅡではコア・メモリ上のユーザ・エリアは96kBである．これだけではNUSSⅡの全機能を持たせることはできないので，コアの重複使用を行っている．最近は仮想記憶方式が主流になり，コア容量をほとんど意識せずにプログラミングができるようになってきたが，BOSⅡにはそれの代りにセグメンテーションという機能があり，LIEDの際プログラム・モジュールをセグメントに分け，実行の際必要に応じて外部記憶からセグメントをコアにロードするという手法で，コアを重複使用することができる．セグメンテーションには次のような規則がある．

- あるプログラムに分岐するとき，そのプログラムの属するセグメントがコア上にないときは，セグメント・ロードをしてから改めてそのプログラムをcallしなければならない．callした方のプログラムは，returnするまでコア上に残っていなければならない．
- サブ・セグメントをロードすると，それに重複するセグメント内のデータは破壊されるので，破壊されてはならないデータは，メイン・セグメント内のcommon areaに置くなどして，コアに常駐させなければならない．新たにロードされたサブ・セグメント内のデータは初期化されている．
- 1つのセグメント内のみから参照されるcommon block および共用ルーチン(組込関数，基本外部関数，GSPなど)はそのセグメント内に置かれる．複数個のセグメント内から共通に参照されるそれらは，セグメント構造図を上位へ(メイン・セグメントに向かって)たどって行って初めて合流するセグメントの中に置かれる．

以上を考慮して，図7のようなセグメント構造とした．"MAIN"はメイン・セグメントで，コアに常駐する．第1階サブ・セグメントのロードはここで行う．"OPEN"には実行開始処理のためのプログラムが入っており，このセグメントは最初に1回だけロードされる．"DIAGRM"はダイアグラムを記述・構成するためのサブルーチン群から成る．"HARDCP"には，GD画面のハードコピーをX-Yプロッタにとるための一連のルーチン(富士通提供)が入っている．"ALALYS"はシステム解析のためのサブルーチン群から成る．さらに，USER'S BLOCK用サブルーチンを使用者が組み込む際のためのメモリの余裕を残す目的で，計算速度を落さない程度に第2階サブ・セグメントを設けた．"ICSET"は解析すべきシステムの初期状態を決定し，画面をANALYSIS PHASEに切り替えるためのルーチンから成る．"STATE"はシステムの状態を時々刻々計算し，結果を表示するためのルーチンから成る．また，"MODIFY"はパラメータ変更指令のときにパラメータを変更し，初期状態を決定し直すためのルーチンから成る．

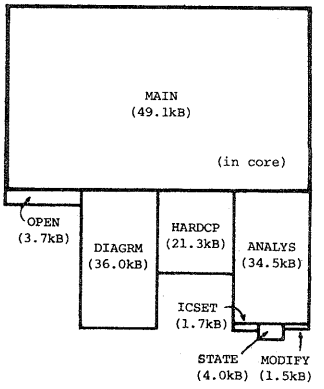


図7　セグメント構造

これらのセグメントの大きさの総計は約152kBになるが，このほかに入出力ルーチンの細かいセグメントが自動的に設けられている．それらはメイン・セグメント内の領域でコアの重複使用が行われている．

おわりに　NUSSⅡのプログラム作成にあたっては，アブノーマルな手法をかなり用いた．たとえば，記憶場所と計算時間の節約のために，サブルーチンの実引数と仮引数の型やサイズを故意に不一致にした部分がある．また，セグメント・ロードでデータが初期化される機能を代入文の代りとして活用した．さらに，ライン・プリンタの行位置が記憶されているアドレスを偶然発見し，配列の添字を故意に逸脱させることによってそれを読み出し，むだな改ページを避けるのに利用した．

近々，当センター第2システムの計算機が入れ替えられることになり，それに伴ってNUSSⅡのプログラムも書き替える必要が生じた．新しいシステムは仮想記憶機能を備え，はるかに使いやすくなるとのことである．しかしBOSⅡも，不便な点は種々ありながら，使い込むほどに興味深いものであり，我々にとってはなつかしいオペレーティング・システムである．

NUSSⅡ開発にあたり，名古屋大学工学部・福村晃夫教授，鳥脇純一郎助教授，同計算機センター・吉田雄二講師，平松敏祐助手，大岸淑徳氏，同プラズマ研究所・津田健三助手はじめ，多くの方々の親切な御指導をいただいた．ここに心からの感謝の意を表する次第である．