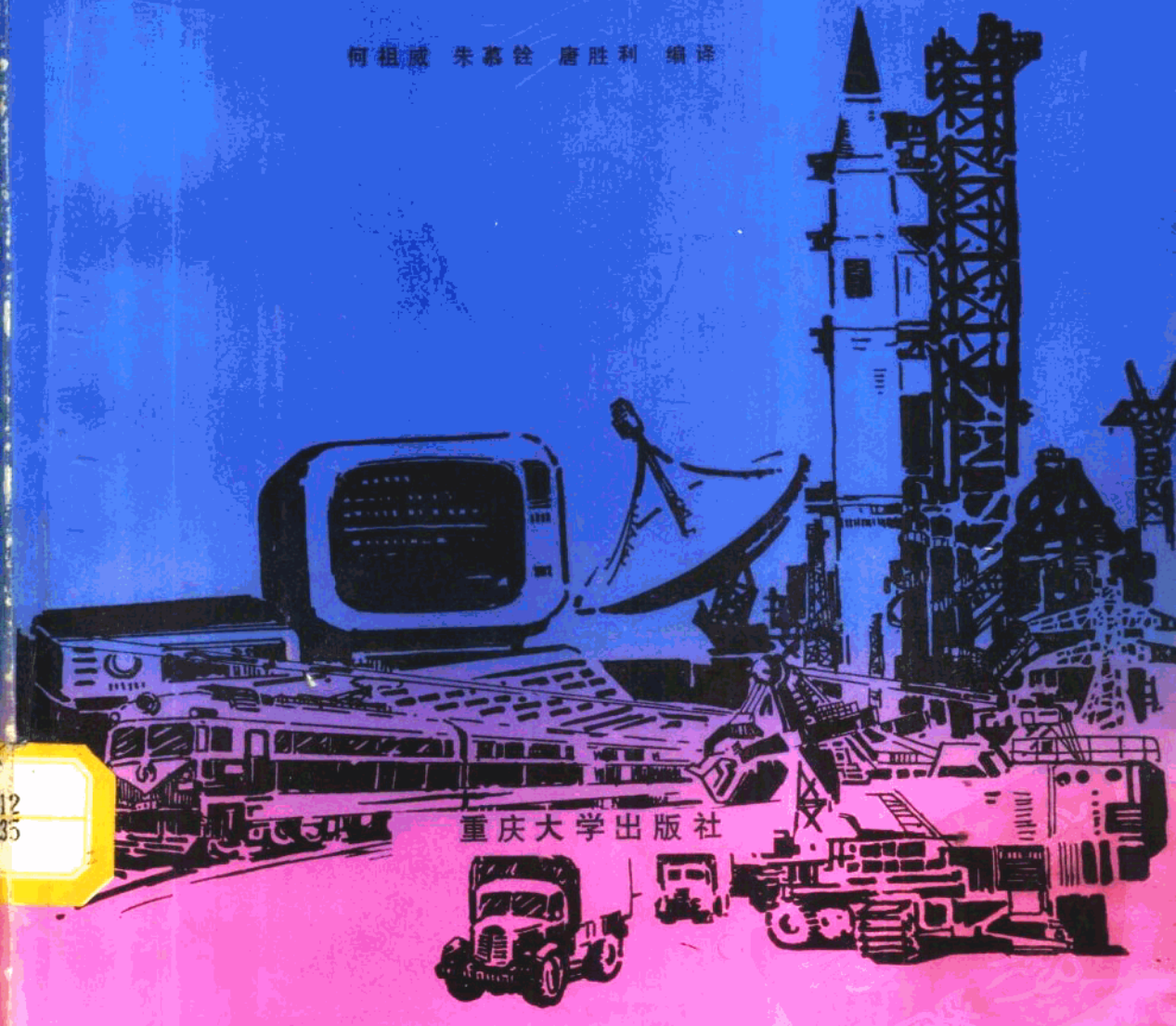
高级连续系统仿真语言 ACSL及其应用

何祖威 朱幕铨 唐胜利 编译



重庆大学出版社

前　言

计算机仿真是一种典型的分析工具，用于在实际系统构造前和构造后，进行对象的设计、计算和特性分析，它包括了建模和用模型作实验的全过程。随着现代科学技术的进步，仿真技术也发展得越来越快，应用范围也越来越广。为了适应仿真技术发展的需要，适用于不同系统和不同领域的仿真语言不断涌现。比如目前国内已介绍过的GPSS、Ada、Dare-P等。本书首次将当今国际上流行最广、功能最强的一种仿真语言ACSL（Advanced Continuous Simulation Language）介绍给国内读者。该语言主要用于模拟和计算用线性或非线性微分方程组描述的连续系统的特性，典型的应用范围是控制系统设计，化工过程的描述，导弹和飞行器仿真，电站动态特性分析，生物和医药模型建立与模拟，以及流体流动和传热分析等。

ACSL是一个面向应用的系统。对广大用户来说，ACSL提供了一种简单明了，易于使用和掌握的动态系统建模工具；对于有经验的仿真工作者和程序员，提供了一种加速程序开发的多功能工具。强有力的面向系统的能力与灵活的交互式的工作方式，使用户将精力集中于模型特性，不必对计算机系统本身赋予更多的注意。特别要指出的是，ACSL所具有的功能极强的宏（MACRO）库，对用户的仿真问题提供了强有力的分析手段，也使用户可能开发自己的仿真模型，以扩充ACSL面向应用的能力。

本书介绍了ACSL的结构、语法规则和功能。

ACSL的基本结构遵循由SCI（Simulation Councils）公司主持的技术委员会所建立的连续系统仿真语言CSSL（Continuous System Simulation Languages）的规范。

本书主要以美国Mitchell and Gauthier公司的ACSL语言参考手册（1981和1987年版）为蓝本编译而成。编译者在尽量保持原手册的体系和风格的基础上，对原手册的章节安排作了适当调整；若干笔误和印刷错误作了订正；结合使用实际，对内容作了部分增删，例题也作了相应变化。

本书由何祖威、朱慕铨、唐胜利编译。其中，何祖威编译第七、八、九、十章、十一章；朱慕铨编译第一、二、三章和附录A、B、C；唐胜利编译第四、五、六章和附录D、E。杨晨、叶佳希、张鹏参加了编译和上机验算工作。全书由何祖威主持编译。

汪瑞芳副教授对本书作了认真审阅，提出了不少宝贵意见，在此谨向她表示诚挚的谢意。

由于编译者水平和经验有限，不足和错误难以避免，望广大读者不吝赐教。

编译者

1990年5月

目　录

第一章 导 言

1.1 ACSL的特点

ACSL是一种面向方程（或传递函数）的语言，对用方程（或传递函数）描述的系统能快速地建模。其表达方式很接近数学语言和自然语言。程序结构简洁明快，易读性好。该语言有几个突出的优点：一是用它建模时间开销少和出错机会减少。它提供了多种功能强的仿真运算函数和数值算法供用户使用，也提供了很好的程序调试手段。二是它的模型与实验分离，便于用户修改模型和设计各种水平、各种目的的仿真实验。三是初始数据、运行中间结果和最终结果、调试信息和出错信息不但可在模型执行中输出，还可以文件形式存放，便于用户事后查询。

ACSL是基于FORTRAN语言的一种仿真语言。图1-1说明了对ACSL程序处理的一般过程。

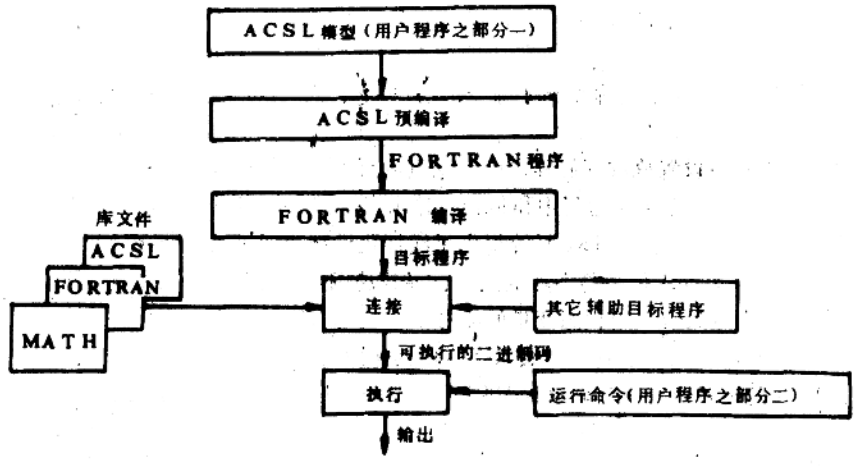


图1-1 ACSL程序处理过程

用户编写的ACSL程序分成两部分（某些类型的机器上表现为写出两个独立的文件），第1部分是模型定义部分（模型文件或称源文件），用于定义被仿真系统的模型或结构；第2部分是执行模型的一系列命令（命令文件），主要用于安排各种仿真实验、设置或改变模型参数、运行时间、控制输出、绘图等等。

与FORTRAN或其它仿真语言相比，ACSL语言有如下一些特点：

1. 连续系统仿真语言主要用于处理微分方程或传递函数表示的系统，因此，ACSL提供了多种数值积分算法供用户任意调用，其中包含求解stiff问题的变步长变阶Gear算法。ACSL允许用户在一个仿真模型的各程序段中采用不同的积分算法及步长，这种对模型的多算法多帧速积分方法，可对快过程和慢过程分别处理，有利于解决stiff问题和提高运算效

率，并且，对每一个状态变量都有独立的误差控制能力。

2. ACSL既可批处理式地运行模型，也可交互式地运行模型并且在终端设备上以图形或数据方式在线显示其中间结果和最终结果。

3. 不同于其它一些仿真语言，ACSL对模型使用的变量、状态变量以及表函数的数目不加限制，它的翻译系统能适应各种大小规格的模型。

4. 用户可以用ACSL以任意的次序书写定义模型的语句，ACSL能自动将这些语句分类排序成可执行的序列，而FORTRAN却必须严格按执行顺序书写。

5. 与CSSL规范相比，ACSL增加了DISCRETE（离散）语句，SCHEDULE（事件进度表）语句和INTERVAL(事件间隔)语句。这些语句的引入，使ACSL能处理诸如对连续域的周期性采样以及将模型计算出的控制作用异步地返回连续域等时间事件，也能处理间断函数等状态事件。

6. ACSL兼容FORTRAN,FORTRAN中的所有库函数或子程序可以被调用。用户也可以在模型中加入自己定义的FORTRAN程序段或其它外部函数。与FORTRAN相比，ACSL在书写格式上更自由，FORTRAN的语句区是7～72列，而ACSL是1～72列。

7. ACSL中除包括常用的类似于FORTRAN的函数和语句外，还具有多种适用于仿真的运算符供用户使用。比如，可变延迟（DELAY),死区（DEAD），偏移（BCKLSH），连续量离散化（QNTZR），积分（INTEG），传递函数（TRAN），……等等。详述见第四、五、六章。

8. ACSL提供了55个运行命令和子命令供用户批处理式或交互式地运行源程序。比如：

SET命令可以重新设置或改变源程序中的变量值而不需重新编译源程序；

PLOT命令及其子命令可在运行中和运行后输出多个模型的多种图形；

ANALYZ命令及其子命令'TRIM'等提供了线性化模型和通过动态修正计算中的状态变量值，使其导数为零来寻求模型稳态点的能力。寻找稳态点是运行模型必不可少的操作；

'FREEZR'子命令具有冻结状态变量的能力；

REINIT命令可将当前状态置为初始状态，使其作为后继仿真运行的出发点；……

总之，ACSL的运行命令充分显示了ASCL的极强仿真运行能力。详述见第七章和第十一章。

9. ACSL还提供了一种MACRO（宏）语言。MACRO语言写成的MACRO块类似于FORTRAN子程序，但比FORTRAN子程序更灵活，用处更大。被定义的MACRO块，既可以象FORTRAN中的子程序一样被调用，也可以象函数子程序一样，作为表达式的一个分量被调用。MACRO中的变量，除类似于FORTRAN中的变量外，还有一种组合变量，它通过虚实结合自动生成一批新变量。比如，定义一个名为MULT的MACRO块如下：

```
MACRO MULT(Y, ID)
      ⋮
Z=Y_ID*T_ID
      ⋮
```

若调用此宏块时，与虚元ID相对应的实元为"1"，则产生了相应的组合变量Y1和T1。这种并置变量的能力为程序的通用化、模块化提供了很大方便。关于MACRO语言详见第八章及第十一章PHYSBE例题。

1.2 简单的ACSL程序介绍

在对此语言作出详细说明之前，先让我们对下列方程进行编程。

$$\dot{x} = y + x(1 - x^2 - y^2)/\sqrt{x^2+y^2};\quad x(0) = 0.5$$

$$\dot{y} = -x + y(1 - x^2 - y^2)/\sqrt{x^2+y^2};\quad y(0) = 0.0$$

所编程序如下所示：

```
R2=X**2+Y**2
X=INTEG(Y+X*(1.0-R2)/SQRT(R2), 0.5)
Y=INTEG(-X+Y*(1.0-R2)/SQRT(R2), 0.0)
```

其中：INTEG是积分运算符，SQRT是平方根函数。

尽管这些语句完全地描述了要求解的方程，但是它并不是一个完整的仿真程序，用ACSL编写的源程序1-1列出了整个运行程序，其中每一个语句均标有数号，它不是源程序的组成部分，仅供说明用。

源程序1-1

```
(1)  PROGRAM LIMIT CYCLE
(2)  CONSTANT XZ=0.5, YZ=0.0, TSTOP=4.0
(3)  CINTERVAL CINT=0.1
(4)  R2=X**2+Y**2
(5)  X=INTEG(Y+X*(1.0-R2)/SQRT(R2), XZ)
(6)  Y=INTEG(-X+Y*(1.0-R2)/SQRT(R2), YZ)
(7)  TERMT(T.GE.TSTOP)
(8)  END
```

运行命令1-1

```
(9)  OUTPUT T, X, Y
(10) START
(11) 'CHANGH INITIAL CONDITIONS AND RUN AGAIN'
(12) SET XZ=0.7
(13) START
(14) STOP
```

为了完成模型的定义，必须有一个PROGRAM语句（1）作为起始；用CINTERVAL语句(3)来定义信息传递的时间间隔；用一逻辑表达式来说明终止仿真运行条件，此表达式为：TERMT语句（7）的自变量；模型的定义必须用END语句（8）作为结束，它与PROGRAM语句相对应。END语句告诉翻译器，后面不允许有任何内容（由FORTRAN语言编写的子程序或函数除外）。要进一步完善的是对方程中的初始值x(0)和y(0)，需分别用一个变量名表示，以及设定运行的终止时间TSTOP，这些由CONSTANT语句（2）来完成。

在程序中，通常会出现一些常量，最好把它们赋给一些变量名，以便在需要时修改。比如，在该程序中就把x(0)和y(0)的初值分别赋给了xz,yz。

除了模型定义语句（源程序）以外，还必须定义一组运行命令（见运行命令1-1），告知怎样实施这个模型。运行命令可以交互地来自一个键盘或一个终端。OUTPUT命令

（9）定义一些变量，这些变量的值在每一个通讯间隔（Communcation Interval）被列出；然后必须用命令START（10）告诉模型运行开始。（11）是一个注释语句，（12）是改变初始值，（13）为重新开始运行。命令STOP（14）是终止仿真研究。在第十一章的例题1中，列出了实际程序，并列出和画出了求解这组方程所获得的结果和图形。

1.3 编程书写规则

ACSL语句行包括80个书写符号，其中前面72个作为语句区，73～80列用于注释语句。ACSL语句可以置于一行的任何位置（前72个），但用户书写的源程序应尽量规范，以使源程序有较清晰的格式。建议采用如下规范：

分段语句——PROGRAM，INITIAL，DYNAMIC，END等起始第1列；

顺序程序段——PROCEDURAL/END语句起始于第6列；

没有标号的语句——起始于第7列；

赋值语句——要确保等号（=）处于第15列或15列后。

读者也可以按照自己的风格，用凹凸形式来编写程序。

在同一行中，可以放两条以上的语句，它们之间用分隔符$分开。如果一条语句要写两行以上，则必须在前一行语句的末端，72列以前任何无空格信息的右侧的任何地方加上一个续行符号（连续3个圆点…，表示下一行是这一行的继续）。再后面的空格可以被排除在含有续行符号的语句之外，继续行的起始空格并不删除。用引号括起来的语句是注释语句，注释语句的内容是由用户自己设定的，但在注释语句中不能用引号（' '）或$作为注释的组成部分。例如：

```
'THIS IS A COMMENT CONTINUED ON TO…
   THE NEXT LINE'
'THE NEXT LINE HAS TWO STATMENTS'
   A=B+C $ X=Y+Z
'COMMENT CAN BE ADDED IN LINE AS…
   A SEPARATE STATMENT'
   X=Y+Z $ 'ASSIGN VALUE TO X'
'NEXT ARE TWO LABELLED STATMENTS'
   LABEL .. C=D+50
   1000 .. FORMAT(1X, 3E12.5)
```

为了使程序醒目，可以在源程序的某些地方插入空行。

第二章 ACSL源程序的结构

2.1 程序结构

ACSL源程序分为隐式结构程序和显式结构程序。如果一个源程序除了PROGRAM…END以外，没有任何其它的段标识符及相应的END语句，则称这个程序为隐式结构程序。(参看源程序2-1)，因为它隐喻整个程序为一个DERIVATIVE段(关于DERIVATIVE段，详见4.1.4)。在进行ACSL翻译时，将对隐式结构中的所有可执行语句进行分类操作(重新排列语句的执行顺序，参见2.2节)，因此，这些语句中，不能循环引用变量，一般不要用IF、GOTO、DO循环等控制转移和循环语句。如果有这类语句，由于重新排序而使程序并不按照用户排定的顺序执行，而无法得到正确的结果。为了使程序更具有灵活性，可以采用3个附加的程序段标识符——INITIAL，DYNAMIC和TERMINAL，并且将模型定义语句置于DYNAMIC段中的一个DERIVATIVE段中。每一个段都必须由一个相应的END语句作为结束标志(参见程序2-1)。如果在PROGRAM…END中，上述4个段标识符中的任何一个出现，则称该源程序为显式结构程序。如果4个段标识符均出现或出现2个以上，则段名必须按下列顺序排列：INITIAL、DYNAMIC、TERMINAL。DERIVATIVE嵌在DYNAMIC…END之中(参见程序2-1)。

程序2-1，显式结构程序的格式

```
PROGRAM
    INITIAL
      ⋮ } 在运行开始前要执行一次的语句，包括登录初始值，
      ⋮ } 注意：此时初始值还没有赋给状态变量。
    END
    DYNAMIC
      DERIVATIVE
      ⋮ } 用于求解系统方程或传递函数的语句，
      ⋮ } 如：INTEG 或 TRAN 等。动态模型
      END
      ⋮ } 每个通讯间隔执行的语句
    END
    TERMINAL
      ⋮ } TERMT 语句中的条件为真时执行的语句
    END
  END
```

2.2 程序流程

当程序开始执行时(发出START命令后)，程序流程就首先进入INITIAL初始化段(图2-1)。

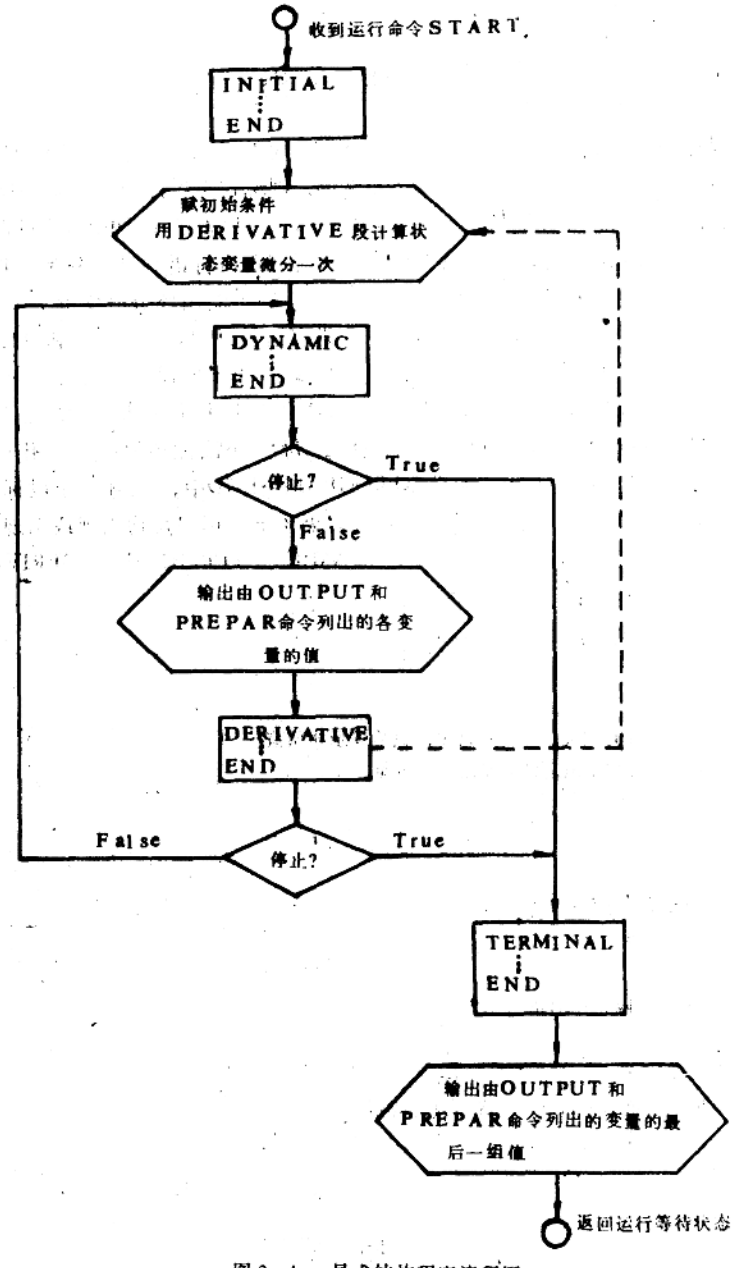


图2-1　显式结构程序流程图

该段在动态段运行之前执行一次运算。通常状态变量（积分运算符的输出）的初始条件在此计算。当在执行INITIAL段的计算时，任何状态变量都不会改变它们的值，因为此时初始条件还没有传递给状态变量。CONSTANT语句不一定放在INITIAL段，因为它们是非执行语句。用户可以把CONSTANT语句放在程序的其它段中。用户要特别小心，不要在该段使用状态变量，因为此时它们自身还没有被定义。然而，已定义的常数或其它已定义的初始条件是可以写在初始化段中的。

将模型初始化的另一种办法是使用RESET运算符。该运算符把已存在的初始条件传递给当前的状态变量。它的使用规定详见4.3.1。

离开INITIAL段后，对积分过程设置初值，它包括将所有初始条件传递给相应的状态变量，以及计算DERIVATIVE段的程序一次。这一过程的作用在于确保在时间为零，存入数据之前，所有要计算的变量都是已知。注意：从所列程序来看，似乎在DERIVATIVE段之前，就执行了DYNAMIC段，事实上并非如此。为DERIVATIVE段中的变量作初始化，并不依赖于DYNAMIC段中的计算。相反，在DYNAMIC段中的计算要用到DERIVATIVE段中计算得到的数值。

再继续看流程图2-1，程序开始执行DYNAMIC程序段中的程序。

对在DYNAMIC段中涉及的变量并无限制。所有状态变量都会有数值，以及在DERIVATIVE段的中间计算将执行完毕。

在这一程序段之后，判断结束条件是否成立，看是否是转入到TERMINAL段。注意：结束条件是由TERMT语句设定的。如果在DYNAMIC程序段中的任意一个TERMT语句中的逻辑表达式为真，则程序此时立即转移到TERMINAL段，并结束程序运行。

如果所有TERMT语句中的逻辑表达式为假，则程序输出在OUTPUT和PREPAR输出变量清单中列出的所有变量的值。OUTPUT是将变量的值在CRT上、打印机上输出，以及存入扩展名为OUT的数据文件。PREPAR是将变量值存入扩展名为RRR的数据文件上，供以后绘图等使用。

现在，积分过程要求在一个通讯间隔期间用嵌入DERIVATIVE程序段中的程序进行积分，以便计算状态变量的导数。

经过通讯间隔后，积分过程以更新了的状态变量返回DYNAMIC段的入口，并再次检查结束条件，在此时，可能由于DERIVATIVE中一条TERMT语句中的逻辑表达式为真，设定了结束标志而使程序结束。如果不结束，则返回，再次执行DYNAMIC段。

如果需要，在各段之间可以用GOTO和标号语句进行控制转移，但不允许从INITIAL、TERMINAL到DYNAMIC段的转移控制，只允许从DYNAMIC段到INITIAL、TERMINAL段的转移控制。同样允许INITIAL段和TERMINAL段之间的转移控制。最后要注意：在DYNAMIC段中至少要有一条TERMT语句或转移控制到TERMINAL段的语句，以使程序循环结束，否则形成“死循环”。

不能进行从DERIVATIVE到其它任何段的控制转移，反之亦然。

当结束条件为真时，程序控制就转向TERMINAL段，并逐句执行段内的语句。程序段执行完后，将读入和处理下一条运行命令（如：PLOT,DISPLY等等）。注意：如果在TERMINAL段中有一个转移控制（GOTO）到INITIAL的语句，那么最后一组要输出的值将不会得到，除非用LOGD运算符记录这一组值。

2.3 程序分类

ACSL翻译器将置于DERIVATIVE段中的模型定义程序进行分类，使得在应用输出量之前先计算出它的值。分类的规则相对而言是较为简单的，这一过程分两步。第一步是检查每一语句，给输出变量作标志以便于计算，并建立该语句所用的输入变量清单。无论在赋值语句或PROCEDURAL的定义头，等号左右两侧可能出现相同的一个变量名，在此情况下，仅仅对左侧（或输出变量）进行分类，使得在应用这一变量以前就确定这个程序段（或语句）的位置。第二步，取出语句的清单，并依次检查它们。如果在一条语句的输入清单上所有变量都没有给它们设定计算标志 则要在翻译文件中加入该语句，并在分类清单上取消输出变量的标志。

如果在一条语句的输入清单上有一个变量作有计算标志，那么存入该语句，并检查下一句。如果在翻译文件中加入了任一语句，那么要重新检查一下所有被存入的语句，再看一看目前能否将它们加入翻译文件和清除它们的计算标志。因为每一个已经处理过的语句的输出变量的计算标志都要被清除，所以这个算法起作用。当然，状态变量在计算时是不作标志的，它的计算必须用积分过程来进行，以及所有的导数值最终要依据状态变量当前的数值来求得。例如：

```
CONSTANT PI=3.142, RZ=1.0
AREA=PI*R**2
R=RZ+LR
LR=INTEG(AREA, 0.0)
```

将第1句转换成FORTRAN中的DATA语句，并建立一个输入/输出清单。没有输入变量且变量PI和RZ有给它们设定的计算标志。

```
DATA PI/3.142/, RZ/1.0/
```

第2句有输入PI和R，而且变量AREA有给它设定的计算标志。第3句有一组输入RZ和LR；对R设定计算标志。最后一句确定了状态量LR和导数AREA。对LR没有设定计算标志，在开始时，就假定它是已知的。

在第一步结束时，符号表如表2-1，设定的计算标志如表中（a）列。

表2-1　　分类时在符号表中的计算标志

符　号	(a) 开始第二步	(b) 以后常数	(c) 以后R=语句	(d) 以后AREA=语句
AREA	ON	ON	ON	
LR				
PI	ON			
R	ON	ON		
RZ	ON			

第二步与有关的I/O清单一起再次开始顺序地阅读语句。第一句没有输入清单，因此可以将它直接输入到翻译文件，这样做时，就清除了PI和RZ的计算标志，表2-1的（b）列。

没有别的未解决的语句了，因而读下一句。输入量是PI和R，当检查时，R仍然是有标

志的，因而存入这一语句。

读下一语句，输入是RZ和LR，这两者都没有计算标志，因而将语句传输到翻译文件，并取消输出变量R的计算标志。再来考察被存入的语句，现在输入变量都无计算标志，因而这一句可以加入翻译文件，并清除AREA的计算标志。

用这种方法能够完全地处理任一适当提出的问题（没有代数循环），翻译程序会将所有语句作出正确的排列。

就分类而言，CONSTANT语句类似于惯用的赋值语句，因而，如果将所有CONSTANT语句置于程序的末端，那么在第一次分类时几乎所有的其它语句都会保持原状，不会被分类。一般认为：好的办法是将它们置于应用它的模块的前面，以减少翻译时间。

在ACSL翻译程序中，是依据下述方法来处理代数循环的，将此代数循环的语句都放在分类程序段DERIVATIVE段的末端。一般认为：代数循环是一个错误，因为一般情况下，增加一个IMPL函数，即可解决代数循环的问题。在此情况下，用将整个代数循环语句串联起来的方法，依次地列出每一语句。通常最容易的是从清单的底部顺着代数循环的语句从后向前看，就会发现在等号（＝）左边的变量会出现在它前面语句的右侧。

例：有如下语句

```
A=B+1
B=C+2
C=A+3
```

这3条语句出现了代数循环的问题，无法正确地对其分类。但是事实上这3条语句可能等价于：

```
A=A+6
C=A+3
B=C+2
```

对于第一条语句，它是一个隐函数方程，因而，应该用IMPL函数来求解（注：上列方程也可能是其它等式，并且，它也可能用一个简单的代数变换，改变方程的形式，得到显函数方程，从而解决这一问题）。

对于顺序程序段（PROCEDURAL…END），ACSL将它作为一个整体来处理（分类时看成一条语句），分类后，其位置在各输入清单上各变量计算出来之后。大多数代数循环是由不正确的PROCEDURAL首部输入清单或者遗漏了状态方程而引起的编程错误。例如：

```
PROCEDURAL (A, B=C, D)
A=C
B=D
END $ 'END OF PROCEDURAL'
D=A
```

第1句意味着A是C，D的一个函数，B也是C，D的一个函数，因此可能会引起一个代数循环的错误。少用和小心地用PROCEDURAL段！用很多个小的PROCEDURAL段比用大的要好。

代数循环的错误可以用代数方法、IMPL运算符或多用PROCEDURAL…END段来解决，从而隐藏了由分类操作定的实际计算顺序，使得能够应用隐含变量先前的数值（需初始化）。

在其它不进行分类操作的程序段中，语句的执行顺序同用户编排的顺序完全一样，各种FORTRAN规定均适用。