5. Ordinary Differential Equations

An equation containing derivatives of a dependent variable with respect to an independent variable, such as

$$\frac{dy}{dx} = xe^y$$

is an ordinary differential equation (ODE). The term "ordinary" refers to the fact that the derivative dy/dx (often written y′) is an ordinary one, not a partial derivative. When no higher order derivatives appear, such as d^2y/dx^2, the equation is a *first order* ODE.

The central problem of integral calculus is the solution of such equations, i.e., determining the function y = f(x) which satisfies the ODE. When a numerical solution is calculated, the curve is determined by a table of values of y as a function of x. In many practical applications, of course, there is more than one dependent variable, resulting in systems of equations of the form

$$y_1' = f_1(x, y_1, ..., y_n)$$

$$\vdots$$

$$y_n' = f_n(x, y_1, ... y_n)$$

When they may be written in this form, where there are only first derivatives and where the functions *f* do not contain derivatives, the equations are called a normal system. Many techniques for the solution of ODEs require that they be reduced to normal systems.

Unfortunately,however, the real world abounds with *implicit* equations such as

$$z' = Az\sqrt{Bz - Cz(y' + z')} = 0$$

$$y' = Ay\sqrt{By - Cy(y' + z')} = 0$$

which cannot be reduced to a normal form. Furthermore, there are often higher order derivatives (y″,y‴,etc.) involved. (When this is the only complication, the system of equations can be reduced to a set of first order equations by introducing new variables and associated equations.)

In FC any of the equations described above may be solved *exactly as they are expressed.* These is no need for reductions, transformations, the introduction of new variables, or any other artifice. In general, there may be any number of equations, they may be linear or nonlinear, they may be explicit or implicit, and they may be of any order.

In order to permit their solution, ODEs must be accompanied by additional informa-

tion in the form of initial or boundary conditions. When these conditions are provided for a single value of the independent variable, e.g., $y(x_o) = y_o$, the problem is called an initial value problem. On the other hand, if y or its derivatives (for higher order ODEs) are specified for two or more values of the independent variable, the problem is called a boundary value problem. It is well known that a unique solution of an nth order initial value problem is determined by specifying n appropriate conditions at one point. This is true in general because the solution of such a problem has n constants of integration, and to provide numerical answers, n initial conditions are needed to determine these constants. If this is a system of m such nth order equations, then a total of (m by n) initial conditions must be supplied.

For the nth order boundary problem, there are also n constants of integration to be determined. In this case, however, for a given set of n boundary conditions, it is possible for the nth order equation to have many solutions or even no solution at all! Thus it is vital to select the boundary conditions carefully, so that the problem to be solved is well-defined and has a unique solution.

The FC ODE solvers are step-by-step numerical integration techniques. A brief description of these solvers is given in Table 5-1. Each time the calculus process is performed, by an INTEGRATE statement, the equations are integrated over a prescribed interval in a direction defined by the sign of the step. Any solver may be used to integrate a single ODE or any number of simultaneous ODEs; and, as shown in Section 5.4, the equations may be of any order.

The basic techniques of the ODE solvers address the solution of explicit, initial value problems. Implicit ODEs and boundary value problems are solved as combined processes, using the ODE solvers in conjunction with the general equation solver AJAX. These techniques are described in Sections 5.5. and 5.6.

5.1. Non-Propagating Solvers

Integration solvers which do not propagate partial derivatives may only be employed in contexts where derivative evaluation is not active. In general, these solvers are far more efficient than solvers which do propagate derivatives. Consequently, they are preferred whenever derivative propagation is not required.

5.1.1. Solver JANUS

JANUS is a fourth order Adams-Moulton predictor-corrector method, employing a fourth order Runge-Kutta starting procedure. The equations are integrated by the Runge-Kutta procedure until enough information has been accumulated to begin the predictor-corrector method, i.e., for three steps. On the fourth step, the technique

TABLE 5-1 Differential Equations Solvers

ATHENA	*Derivative propagating, variable-order Runge-Kutta method with output variable limiting.* ATHENA is a single step (nonmemory) integration method which applies the Runge-Kutta method with orders 2,3,4, and 5. As a nonmemory solver, it does not require reinitialization whenever discontinuities in the differential equations are present. Limits may be imposed upon the dependent variables of integration, and partial derivatives may be propagated through the integration process, allowing nesting of integration within optimization processes.
GEMINI	*Hybrid (propagating/nonpropagating) rational-function approximation method.* GEMINI is a hybrid solver which automatically operates as MERLIN when partial derivative propagation is required, and operates as NEPTUNE otherwise.
ISIS	*Derivative-propagating, fourth-order Runge-Kutta-Gill method.* ISIS is a memory solver designed as a recursive function allowing the solution of nested differential equations. Partial derivatives may be propagated through the integration process, allowing nesting of integration within optimization processes.
JANUS	*Nonpropagating, fourth-order Adams-Moulton method.* JANUS is a memory method with an automatic procedure for varying the integration step in order to improve execution efficiency by taking the largest possible step for a given error criterion. It does not propagate derivatives through integration, thus it may not be nested within an optimization process.
JANISIS	*Hybrid (propagating/nonpropagating) fourth-order Runge-Kutta Adams-Moulton method.* JANISIS is a hybrid solver, which automatically operates as ISIS when partial derivative propagation is required and operates and JANUS otherwise.
MERCURY	*Nonpropagating, optimal step size and stiff equation method of Gear.* MERCURY offers two predicator-corrector methods, an Adams method and a special formulation by Gear for the solution of stiff differential equations. MERCURY chooses both the order and the step size in order to minimize problem solution time while holding truncation error within prescribed limits. MERCURY does not propagate derivatives through integration, thus it may not be nested within an optimization process. However it does activate the evaluation of partial derivatives within its model, and uses the values of partial derivatives in the integration process.
MERLIN	*Derivative propagating, rational-function approximation method.* MERLIN employs the memory technique of Gragg, Bulirsch and Stoer. It optimizes both the step size and the order of approximation so as to minimize the computation necessary to satisfy a specified error bound. It propagates partial derivatives through integration, allowing nesting of integration within optimization processes.
MINERVA	*Nonpropagating, variable-order Runge-Kutta method with state variable limiting.* MINERVA is identical to ATHENA except that it does not propagate derivatives through integration. Thus it may not be nested within optimization processes.
NEPTUNE	*Nonpropagating, rational function approximation method.* NEPTUNE is identical to MERLIN except that it does not propagate derivatives through integration. Thus it may not be nested within optimization processes.
PEGASUS	*Nonpropagating, fifth-order, Sarafyan-Runge-Kutta method with state variable limiting.* PEGASUS employs a fifth order nonmemory Runge-Kutta technique known as Sarafyan imbedding which is used to optimize step size for a given accuracy requirement. Limits may be imposed upon the state variables of integration. However, partial derivatives are not propagated through integration, thus PEGASUS may not be nested within an optimization process.

switches automatically to the Adams-Moulton method.

During the Runge-Kutta process, the differential equations are evaluated four times per step by executing the model four times. Once the Adams-Moulton procedure begins, the model is executed only two times per step, although the accuracy of the results is the same as for the Runge-Kutta steps.

JANUS cannot be "nested" within a calculus process which activates derivative evaluation; however, its model may include any calculus process except another integration involving JANUS. For example, the differential equations may be implicit equations (see Section 5.5.) or integro-differential equations using the definite integral functions (INTEGRAL, ROMBINT, etc.).

A major advantage of JANUS is that it employs an automatic procedure for varying the integration step in order to hold numerical error within prescribed limits. When the controls associated with this procedure are properly set, the step size will be either increased or decreased in a manner which permits the largest possible step within defined error criteria.

Initiating JANUS - The INITIATE statement for JANUS takes the following form:

INITIATE JANUS {(*controller*)}; {{**WITH**}**FLAG** *signal*;}
FOR *model*; **EQUATIONS** *ode/vars*; **OF** *ind-var*; **STEP** *delta*; **TO** *limit*

The meaning and form of the symbols: *controller*, *model*, *ode/vars*, *ind-var*, *delta*, and *limit* are identical to those described for integration processes in Sections 1.3 and 2.2.1

The equations are identified by the *ode/vars* list which specifies the names used in the *model* to represent the state variables and their derivatives. This list has the form

der/var,der/var,...

where *der* identifies ordinary derivatives and *var* identifies the corresponding dependent variables. For more detailed description, see Section 2.2.1.

The scalar variable *signal* is set by JANUS to denote its action with respect to step size control. After each INTEGRATE statement, JANUS sets *signal* as follows:

Value	Meaning
0	The step size was either increased or unchanged.
n	The step size was decreased because of excessive error in the nth ODE[1].

1 The equations are numbered by their sequence in the *ode/vars* list, not by their order of computation in the model.

The optional parenthesized phrase {(*controller*)} performs the usual function of identifying a controller so that control values may be changed for this use of JANUS.

The model must be a MODEL label, identifying the calculus model for this process. This model must compute the ODEs and any auxiliary values which they require. The MODEL may not have parameters, but may call other MODELs or FMODELs that do.

Executing JANUS - Once the INITIATE statement has been executed, function generation is performed by JANUS whenever the statement

INTEGRATE *model*; **BY JANUS**

is executed. This statement generates the integral function up to the limit value of *ind-var*.

This statement generates no printed output. If the values at the end of a step are required, they must be printed by ordinary FORTRAN output statements.

Example

If the integration was initiated by

```
X = 0 :Y = 1: T = 0 : DT = 0.01
INITIATE JANUS; FOR DIFF; EQUATIONS XDOT/X, YDOT/Y;
    OF T; STEP DT; TO TF
```

then integration up to T = 1, with printing at .1,.2,.3,... may be performed by

```
TF = 0
DO WHILE (TF.LT.1)
   TF=TF+.1
   INTEGRATE DIFF; BY JANUS
END DO
```

If the values of any of the dependent variables or the independent variable are altered after the INITIATE statement, this imposes a discontinuity in the equations. In fact, this really defines a new initial value problem. While this is perfectly valid, it is mandatory that an INITIATE statement be executed once more before continuing integration. This is necessary because the Adams-Moulton procedure maintains a "memory" of previous steps. On the other hand, the step size may be modified at will between INTEGRATE statements. (If this is done, JANUS will automatically reimpose its Runge-Kutta starting procedure in order to refresh its memory.)

Controlling JANUS - JANUS has four control variables, all associated with its automatic step size adjustment. They are as follows:

Variable	Description	Preset Value
ERRLO	**Fractional error lower limit.** If the relative predictor-corrector errors[2] for all equations are less than this value then the step size is doubled for use on the next step. The upper limit is STEPMAX.	10^{-7}
ERRHI	**Fractional error upper limit** If any equation generates a relative error greater than this value, the step size is halved and the integration step is reexecuted at the lower value. This may occur repeatedly during one step if the resulting error is still too large. The lower limit is STEPMIN.	10^{-4}
STEPMAX	**Maximum step size.** The preset value is effectively no limit.	10^{9}
STEPMIN	**Minimum step size** The preset value allows the adjustment control to be effectively free to reduce the steps. If no real limit is imposed, it is desirable to test the step size periodically while integrating in order to see if it is decreasing unreasonably. Often this is a good indication of a poorly posed problem.	10^{-9}

2 The relative error is the absolute error divided by the magnitude of the dependent variable.

5.1.2. Solver MERCURY

MERCURY applies Gear's method[3] for integration of ordinary differential equations. As with other FC integration solvers, MERCURY is suitable for both linear and nonlinear systems of arbitrary number and degree. It has, however, two unique capabilities as described below.

- *Stiff ODE Systems* - An ODE system is termed stiff when it contains greatly differing time constants or oscillation frequencies. Conventional integration techniques are impractical for such systems because they require inordinately small step sizes to achieve acceptable accuracy and stability. MERCURY offers two predictor-corrector algorithms, an Adams method and a special formulation designed by GEAR for the solution of stiff systems.
- *Optimal Step Size* - Regardless of which algorithm is selected, MERCURY chooses both the order and step size so as to minimize problem solution time while holding truncation error within prescribed limits. For the Adams method, the order may be from 1 to 13; for the stiff method it may range from 1 to 6. The gain in efficiency depends upon the length of the integration span and the character of the equations. However, improvements of one or more orders of magnitude are not unusual.

Derivative Evaluation - MERCURY automatically activates the evaluation of partial derivatives with respect to the dependent variables of integration.[4] If these variables are changed by FC statements in the model, a warning diagnostic will be issued indicating that a change in an independent variable (of derivative evaluation may invalidate existing partial derivatives.

Initiating MERCURY - The INITIATE statement for MERCURY takes the general form

INITIATE MERCURY {(*controller*)}; **FOR** *model*; **EQUATIONS** *ode/vars*; **OF** *ind-var*; **STEP** *delta*; **TO** *limit*

The meaning and from of the symbols: *controller*, *model*, *ode/vars*, *ind-var*, *delta* and limit are identical to those described for integration processes in Sections 1.3 and 2.2.1.

Executing MERCURY - The integral function is generated by MERCURY whenever

---

3 A brief description is given in: Gear, C.W., "The Automatic Integration of Ordinary Differential Equations," *Communications of the ACM* 14,3 (Mar 1971), 176-179. A more comprehensive discussion may be found in: Gear, C.W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, 1971.

4 This means that MERCURY cannot be nested inside another calculus process.

the statement

INTEGRATE *model*; **BY MERCURY**

is executed. Partial derivative evaluation is activated and a sequence of integration steps is performed until *ind-var* reaches the value specified by *limit*. During integration, the step size will be modified to minimize the amount of computation required. However, the value of *delta* will not be changed. Rather, this value is used merely as an initial estimate. To obtain the best results, the value of *delta* should be considerably smaller than the expected average step size, perhaps by one or two orders of magnitude.

Terminating MERCURY - When integration has reached the limit, partial derivative evaluation is deactivated and execution continues following the INTEGRATE statement. At this point, results may be analyzed or displayed, further integration may be performed, or the integration process may be ended by

TERMINATE *model*

Controlling MERCURY - MERCURY employs the following control variables:

Variable	Description	Preset Value
MAXERR	**Relative error limit.** The order of approximation and the step size are adjusted to assure that the largest relative truncation error does not exceed MAXERR, i.e., max(abs(error(y)/y)) < MAXERR	10^{-4}
STIFF	**Option to select integration algorithm**	**0**
	Value — *Option*	
	=0 — Selects the Adams method. This is more accurate and, in general, more efficient for non-stiff ODEs.	
	≠0 — Selects the stiff method. This method is suitable for both stiff and non-stiff systems. Where there is doubt, it should be chosen.	
SAVE	**Option to save generated points in a file.** When set non-zero, points of the generated function are saved, at increments of *ind-var* equal to +SAVE over the specified interval, for subsequent analysis or display.	**0**
STEPMAX	**Maximum step-size.** The preset value effectively specifies no limit. Sometimes an upper limit is necessary to preserve stability of approximation.	10^{9}
STEPMIN	**Minimum step-size** If error evaluation requires a step smaller than this value, integration terminates with a suitable diagnostic. Running time and round-off concerns may influence the use of a lower limit.	10^{-9} x *Delta*

Retrieval of the Integrated Functions - Because step-by-step integration is impractical for MERCURY, intermediate values of the generated functions are not readily accessible. This restriction is minimized by activating the SAVE control variable. When this is done, function values are saved on a scratch file (unit 98) during integration and may be retrieved after the INTEGRATE statement by reading the values from the file as follows:

READ(98) *ind-var, variable list*

The data saved by MERCURY are:

Index	Data
1	Independent variable vector
2 ... n+1	Vectors of the n dependent variable functions from left to right in the *ode/vars* list
n+2 ...2n+1	Vectors of the relative truncation errors for the n dependent variable vectors[5]
2n+2	Vector of maximum value of above error vectors[6]
2n+3	Vector of current step sizes
2n+4	Vector of cumulative average step sizes[7]
2n+5	Cumulative number of steps

Example

A particularly stiff set of ODE's

$$\dot{z}_i = - B_i z_i + z_i^2 \qquad i = 1,2,3,4$$

has been proposed by F.T. Krogh, for testing stiff equation methods, where the B_i are the constants -10, .001, 800, 1000. The general solution to this system is

$$z_i - B_i/[1 + C_i \exp(B_i t)]$$

If the initial values are $z_i(0) = -1$ then $C_i = -(1 + B_i)$

This problem is phrased in FC as follows:

---

5 These are only proportional to the actual errors, but provide useful indication of which equations are contributing the most error.

6 Useful in determining the most critical portions of the problem.

7 This will be a very conservative measure since it will normally be very heavily weighted by early small steps.

```
      PROBLEM KROGH(5000,1000,1000)
          COMMON /KROG/ Z(4),ZDOT(4),T,DT,TF
          Z(1)=-1.0 : Z(2)=-1.0 : Z(3)=-1.0 : Z(4)=-1.0
          T=0.0 : DT=.5 : TF=1.0
          INITIATE MERCURY(SETUP); FOR STIFF;
              EQUATIONS ZDOT/Z; OF T; STEP DT; TO TF
          INTEGRATE STIFF; BY MERCURY
          REWIND 98
          PRINT 5
    5     FORMAT('       TIME            Z1            Z2')
          DO 10 I=1,100
              READ(98) TIME,Z1,Z2,Z3,Z4
              PRINT 6,TIME,Z1,Z2
    6         FORMAT(3E14.6)
   10     CONTINUE
      END
      CONTROLLER SETUP(MERCURY)
          ERRMAX=1E-6
          STIFF=1
          SAVE=0.01
      END
      MODEL STIFF
          COMMON /KROG/ Z(4),ZDOT(4),T,DT,TF
          DIMENSION B(4)
          DATA B/-10,0.001,800,1000/
          DO 10 I=1,4
              ZDOT(I)=Z(I)*(Z(I)-B(I))
   10     CONTINUE
      END
```

OUTPUT:

```
     TIME            Z1            Z2
0.100000E-01 -0.109367E+01 -0.990089E+00
0.200000E-01 -0.119495E+01 -0.980373E+00
0.300000E-01 -0.130423E+01 -0.970845E+00
0.400000E-01 -0.142189E+01 -0.961501E+00
0.500000E-01 -0.154828E+01 -0.952334E+00
0.600000E-01 -0.168370E+01 -0.943341E+00
0.700000E-01 -0.182840E+01 -0.934516E+00
0.800000E-01 -0.198257E+01 -0.925855E+00
0.900000E-01 -0.214632E+01 -0.917352E+00
0.100000E+00 -0.231969E+01 -0.909004E+00
0.110000E+00 -0.250260E+01 -0.900807E+00
0.120000E+00 -0.269487E+01 -0.892756E+00
0.130000E+00 -0.289621E+01 -0.884847E+00
0.140000E+00 -0.310619E+01 -0.877078E+00
0.150000E+00 -0.332428E+01 -0.869443E+00
0.160000E+00 -0.354979E+01 -0.861940E+00
0.170000E+00 -0.378194E+01 -0.854566E+00
      :             :             :
      :             :             :
      :             :             :
0.940000E+00 -0.999255E+01 -0.515109E+00
0.950000E+00 -0.999326E+01 -0.512466E+00
0.960000E+00 -0.999390E+01 -0.509849E+00
0.970000E+00 -0.999448E+01 -0.507259E+00
0.980000E+00 -0.999500E+01 -0.504695E+00
0.990000E+00 -0.999548E+01 -0.502156E+00
0.100000E+01 -0.999591E+01 -0.499642E+00
```

5.2. Propagating Solvers

Integration solvers which propagate partial derivatives may be employed in any context in a FC program. However, as derivative propagation requires significant computation overhead, these solvers are not recommended for use in contexts where derivative evaluation is not active.

5.2.1. Solver ISIS

ISIS is a fourth order Runge-Kutta method, employing the Gill technique for error control. In general, Runge-Kutta methods are known as "single-step" or "self-starting" techniques, since each step is independent of the last, just as though each represented a new initial value problem. The Gill modification considerably reduces numerical error, but it does so by "remembering" error correction terms as the integration progresses. Thus, the technique represented by ISIS must be viewed as having a memory, analogous to the predictor-corrector memory of solver JANUS.

The calculus model, representing the differential equations for ISIS, may contain any calculus processes, including integrations involving ISIS. There are no controls for ISIS and it employs no parameters.

Initiating ISIS - The INITIATE statement for ISIS has the following form

INITIATE ISIS; FOR model; **EQUATIONS** *ode/vars*;
OF *ind-var*; **STEP** *delta*; **TO** *limit*

The meaning and form of the symbols *model*, *ode/vars*, *ind-var*, *delta* and *limit* are identical to those described for integration processes in Sections 1.3 and 2.2.1

Examples

INITIATE ISIS; FOR MOD2; EQUATIONS A/B, C/D;
OF INDEP; STEP INC; TO XFINAL

INITIATE ISIS; FOR DEQS; EQUATIONS VECD/VECV;
OF ZZ; STEP DELZZ; TO ZZEND

Executing ISIS - Once the INITIATE statement has been executed, the integration may be performed by execution of:

INTEGRATE *model*; **BY ISIS**

where *model* is the MODEL label identified in the INITIATE statement.

As with all step-by-step calculus processes, ISIS produces no printed output. If values

at the end of a step are required, they must be generated by ordinary FORTRAN output statement. See the example in Section 5.1 for an approach to step printing.

Because of the Gill modification, ISIS must be reinitiated whenever the value of any dependent variable or the independent variable is modified. Failure to do so will disable the error control mechanism and, in fact, may cause it to be counterproductive. It is doubly critical to do this in circumstances in which derivative evaluation is in progress, since the resulting discontinuities in derivatives may have disastrous consequences. In particular, when ISIS is nested within a calculus process which activates derivative evaluation, such as AJAX, the INITIATE and INTEGRATE statements must both be placed within the nested model. Examples of proper usage in the case of combined calculus processes are given later in this section.

Terminating ISIS - When the integration process has reached the desired solution, it may be terminated by

TERMINATE *model*

This terminates ISIS for the model being integrated. (Other models which ISIS might be integrating remain unaffected.)

The significance of terminating ISIS is that the integration variables are released from preemption, the model is freed for use in other calculus processes, and dynamic storage used by the process is released. When ISIS is nested within an iterative process, TERMINATE must be used to prevent the multiplicative use of dynamic storage.

5.2.2. Solver JANISIS

JANISIS is a hybrid solver which executes identically to solver ISIS when derivative evaluation is active and identically to the solver JANUS otherwise. In all respects, the descriptions of ISIS and JANUS apply to JANISIS. In particular, the control variables of JANISIS are those of JANUS (ISIS has no control variables.)

This solver is more efficient than ISIS when nested within calculus processes employing solvers (such as AJAX) which execute their models with derivative evaluation both active and inactive.

5.3. The Integration Model

The calculus model representing the ODEs must contain both the equations themselves and any auxiliary calculations required for their computation. These equations may be computed in any logical sequence. In addition, they may be computed either in the MODEL procedure or in any subsidiary MODEL or FMODEL which it executes.

However, the MODEL referenced in the INITIATE and INTEGRATE may not have parameters.

While it may be obvious, it should be noted that the integration variables used by the model must be global variables, since they are shared by the model, the solver, and the program unit in which the calculus process appears. It may be necessary, therefore to include these variables in appropriate common blocks.

Example

Consider a projectile drag force F_D which is a function of the projectile velocity V, according to the following formula

$$F_D = C_D \rho V^2/2$$

If F_D is employed in the differential equation

$$m\, dV/dt = -F_D \cos\theta$$

then the equation for F_D must be included in the model block as an auxiliary equation. Thus

```
MODEL PROJECT
   COMMON/DRAG/FD,DVDT,CD,RHO,V,THETA,AMASS
    FD = CD*RHO*V**2/2
  DVDT = - FD*COS(THETA)/AMASS
END
```

In general, the model will be executed more than once per integration step. Thus, it is desirable to exclude from the model all calculations which are not essential to the ordinary differential equations themselves.

5.4. Higher Order Differential Equations

The integration of higher order ODEs (second order, third order, etc.) may be accomplished with any integration solver without reducing the equations to a normal system.

As a basis for discussion, consider a set of two simultaneous first order equations:

$$dx/dt = x + y$$

$$dy/dt = x - y$$

Choosing obvious names, the INITIATE statement for this problem might be:

```
INITIATE ISIS; FOR TWO EQUATIONS XP/X, YP/Y;
OF T; STEP DT; TO TF
```

Thus, XP is specified to be the first derivative of X with respect to T:

$XP = dX/dT$

Now suppose that, instead, there was the single second order equation

$d^2x/dt^2 = 2x$

then, clearly, there must be an integration variable to represent the second derivative, which might be names XPP:

$XPP = d^2X/dT^2 = d/dT\,(dX/dT) = dXP/dT$

Thus, to fully specify the variables for this equation in an INITIATE statement, it must take the form:

```
INITIATE ISIS; FOR DOUBLE EQUATIONS XPP/XP,XP/X;
OF T; STEP DT; TO TF
```

where, by analogy, XPP is interpreted as the first derivative of XP with respect to T and XP is interpreted as the first derivative of X with respect to T. Note that the form of the specifications requires a name for XP even though it may not actually appear in the second order equation.

The preceding discussion may be generalized as follows. For each *n*th order equation, the *n* derivatives must all be assigned names and there must be an entry in the ode-vars list for each one.

Example

The fifth order equation:

$d^5y/dt^5 = 3t^2 + y^3$

may be integrated be first initiating JANUS by:

```
INITIATE JANUS; FOR ORDER5
EQUATIONS D5Y/D4Y,D4Y/D3Y,D3Y/D2Y,D2Y/DY,DY/Y;
OF T; STEP DT; TO TF
```

and then by executing

```
INTEGRATE ORDER5; BY JANUS
```

using the model

```
MODEL ORDER5
    COMMON/DERS/DSY,D4Y,D3Y,D2Y,DY,Y,T
    D5Y = 3*T**2 + Y**3
END
```

As usual with the integration solvers, all dependent variables must be set to their initial values before the INITIATE statement is executed. In the above example,

D4Y,D3Y,D2Y,DY, and Y must all be initialized, in addition of course to T and DT. (As noted in the introduction to Section 5, a fifth order initial value problem requires five initial conditions to specify a unique solution.)

Where the problem involves simultaneous equations, of any order, the same approach is used to specify all of the pertinent derivatives and variables. For three simultaneous equations, of first, second, and third order respectively, there would be a total of six pairs of derivative/variable sets in the *ode-vars* list.

Example

For the simultaneous equations

$d^3y/dt^3 = xy$

$d^2x/dt^2 = xt$

The INITIATE statement could be written

```
INITIATE ISIS FOR MIXED;
EQUATIONS D3Y/D2Y, D2Y/DY, DY/Y, D2X/DX, DX/X;
OF T; STEP DT; TO TF
```

All of the preceding examples illustrate a systematic specification of the higher order derivatives: they appear in sequence, from high to low. This is not optional. The integration solver requires this discipline in order to perform its internal reduction.

If the variables in the list are vectors, then each element of a higher order derivative vector must be the first derivative of the corresponding element of its succeeding vector, in complete analogy to the scalar form.

5.5. Implicit Differential Equations

An implicit ordinary differential equation cannot be reduced to the normal form

$y' = f(x, y)$

but rather has the general form

$g(x, y, y') = 0$

Systems of such equations, which may involve higher order derivatives, are solved in FC by a combined calculus process. The integration process is executed exactly as if the model involved explicit equations. Within the model, however, the equations are actually calculated by a general algebraic equation solver, such as AJAX.

As far as the specification of the integration process is concerned, there is absolutely no difference between the solution of implicit and explicit equations. In a given model, in fact, some equations may be implicit while others are explicit. The calculus model

itself, of course, is quite different, reflecting the different character of the mathematics.

Any FC integration solver may be used to solve implicit equations and all of the preceding discussions of integration apply to such equations. The preferred solvers, however, are JANUS and MERCURY since the fact that they execute the model only twice per integration step provides a very significant computational efficiency.

The procedure involved in solving this combined process is best described in terms of the following representative example. For descriptions of the solvers employed, refer to appropriate sections of this manual.

Example

The implicit differential equations:

$$dx/dt + [1+(dx/dt + dy/dt)/(x+y)]^{1/2} = 0$$

$$dy/dt + [1+(dx/dt + dy/dt)/(x+y)]^{1/2} = 0$$

may be solved by the following program

```
PROGRAM IDE
  COMMON/IMP/DXDT,X,DYDT,Y
  READ *,X,Y,DYDT,DYDT ! Initial conditions & initial guesses
  READ *,T,DT,TPRINT,TFINAL
  INITIATE JANUS; FOR DEQ;
    EQUATIONS DEDT/X,DYDT/Y; OF T; STEP DT; TO TF
  TF = TPRINT 10
  DO WHILE (TF.LT.TFINAL)
    INTEGRATE DEQ; BY JANUS
    PRINT *,T,DXDT,X,DYDT,Y
    TF=TF+TPRINT
  END DO
END

CONTROLLER SET (AJAX)
  SUMMARY=0
END

MODEL DEQ
  COMMON/IMP/DXDT,X,DYDT,Y
  FIND DXDT,DYDT; IN IDE(GX,GY); BY AJAX(SET); TO MATCH GX,GY
END

MODEL IDE(GX,GY)  ! IMPLICIT EQUATIONS
  COMMON/IMP/DXDT,X,DYDT,Y
  RAD=SQRT(1+(DXDT+DYDT)/(X+Y))
  GX=DXDT+RAD
  GY=DYDT+RAD
END
```

5.6. Rational Function Extrapolation (RFE) Solvers

The following three solvers, NEPTUNE, MERLIN, and GEMINI employ the technique of rational function extrapolation developed by Gragg, Bulirsch, and Stoer:

- Bulirsch and Stoer, "Numerical Treatment of Ordinary Differential Equations by Extrapolation", Numerische Mathematik 8, 1-13 (1966).

This method, like that of MERCURY, optimizes both the step-size and the order of approximation so as to minimize the computation necessary to satisfy a given error bound. These solvers offer significant advantages in controlling the local truncation error and in determining an appropriate step-size dynamically over the range of integration, and can provide considerable time savings for lengthy integrations.

In contrast to MERCURY, however, the RFE solvers also provide the following benefits:

- They do not invoke the evaluation of partial derivatives.
- They usually require fewer model evaluations when very precise results are mandated. For error bounds smaller than about 1.0E-8, the savings over Runge-Kutta or Adams methods can be from a factor of 2 to as much as 30. While MERCURY provides similar advantages, as the error bound drops the superiority of RFE becomes more pronounced.
- They are self-starting so that the full power of the method is applied over the entire range of integration.

The RFE solvers are memory solvers, i.e. each INTEGRATE statement uses information generated at a preceding INTEGRATE (if any). Thus, a change in any of the key variables except the step-size and integration limit demands re-initiation of the solver. For example, if a differential equation has a discontinuity then the solver must be halted and re-initiated at this point. (This is a restriction for all memory solvers, i.e. for all integration solvers except MINERVA, ATHENA, and PEGASUS.) When there are numerous discontinuities, PEGASUS offers a better method for error control and step-size management.

The three RFE solvers NEPTUNE, MERLIN, and GEMINI are distinguished only in the way they operate in combination with other calculus processes:

- NEPTUNE is a straight FORTRAN implementation of the RFE algorithm, which does not propagate partial derivatives
- MERLIN is an overloaded implementation of the RFE algorithm designed to propagate partial derivatives
- GEMINI is a hybrid of the two designed to operate as MERLIN when partial derivative evaluation is active and to operate as NEPTUNE when it is not.

5.6.1. Solver NEPTUNE

The following description of NEPTUNE applies equally to MERLIN and GEMINI.

Initiating NEPTUNE - The INITIATE statement for NEPTUNE takes the form:

INITIATE NEPTUNE {*(controller)*}; **FOR** *model*; **EQUATIONS** *ode/vars*; **OF** *ind-var*; **STEP** *delta*; **TO** *limit*

The meaning and usage of the elements of this statement are identical to those described for MERCURY in section 5.1.

Contolling NEPTUNE - NEPTUNE has three control variables, all concerning step size control, as follows :

Variable	Description	Preset Value
ERRMAX	**Relative error limit.** The order of approximation and step-size are adjusted to assure that the largest relative local truncation error does not exceed ERRMAX, i.e. MAX(ABS(YERROR/Y) < ERRMAX	**0.0001**
STEPMAX	**Maximum allowed step-size** The preset value provides effectively no limit. If the equations are potentially unstable, some appropriate upper bound on the step is advisable.	**10^9**
STEPMIN	**Minimum allowed step-size.** This control is provided as a protection against modeling errors or a choice of ERRMAX which is incompatible with the differential equations. If error control requires a step smaller than this value, integration halts with a fatal error diagnostic. Do not use this control in an attempt to disable step-size management.	**10^{-9}**

The following considerations concerning the step-size, *delta*, should be noted :

- NEPTUNE adjusts the value of *delta* after completing execution of the INTEGRATE statement so that subsequent integrations may proceed from an optimal step-size. Interference with this judgment by altering the value of *delta* is allowed but is discouraged.
- The significance of the initial value of *delta* is quite different from the cases of MERCURY and PEGASUS, both of which apply unique strategies for improving the step-size. In the case of PEGASUS, the initial value is considered to be valid estimate and any adjustments are relatively conservative. MERCURY treats the initial value as a tentative lower limit for its iterative improvement and thus the value is not too critical and should always be selected on the low side (perhaps by 1 or 2 orders of magnitude).