

CHAPTER 15 MODELING AND SOLVING DAE PROBLEMS IN THE ASCEND SYSTEM

Ex: Heating a tank
filling with liquid

In this chapter we discuss using ASCEND in the modeling and solving of initial value problems that involve both ordinary differential and algebraic equations (DAEs). An example of such an initial value problem is to model the heating of a tank filling with liquid. In solving, one first establishes an initial condition at time zero, such as for the tank being at 300 K, 25% full, and at 1 atm, and then marches forward in time until one reaches a stopping condition, such as the first to happen of either ten minutes has passed, the tank is 75% full or the vessel temperature reaches its boiling temperature.

15.1 PREVIOUS IVP MODELING AND SOLVING IN ASCEND VS THE NEW APPROACH

Many available programs exist to solve initial value ordinary differential equation (ODE) problems, e.g., LSODE. Chapter 12 in this series of HowTo chapters on ASCEND discusses our attaching of LSODE as a solver package to ASCEND to solve differential-algebraic equation (DAE) problems. Figure 15-1 illustrates. In this previous approach, we used ASCEND to solve the algebraic equations (the top box) whenever LSODE (the lower box) requested it as it iteratively solved the integration equations it used to step forward in time. As far as LSODE is concerned, the model is an ODE model, as ASCEND has effectively eliminated the algebraic variables by solving for them “out of the sight” of LSODE. In effect for each time step, the top box received values of the state variables from LSODE, used the model in

ASCEND to solve for the algebraic variables and the resulting derivatives of the states, and passed the latter back to LSODE. LSODE then repeatedly requested values for the states until the computed derivatives and states it then computed stopped changing, at which time it commenced computations for the next time step.

Inefficient as there is an outer loop iterating over an inner loop

Such a solution procedure is inherently inefficient as it contains an outer loop (LSODE) repeatedly calling an inner loop to converge the model equations. If the outer loop takes five iterations and the inner three per step, ASCEND is solving the model equations 15 times per step. It would be much more efficient to use ASCEND to solve both the model and integration equations at the same time, with the hope that it might only require three model solutions (or fewer) per time step.

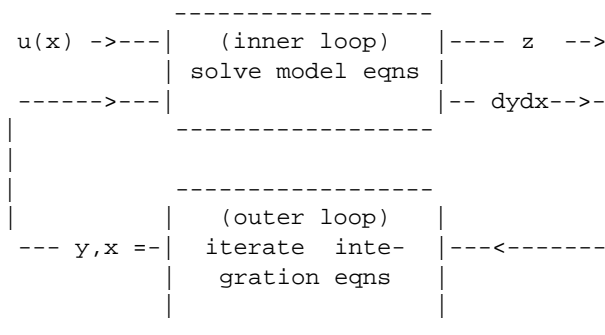


Figure 15-1 Typical IVP ODE solving approach (x is the independent variable -- e.g., time, and y, u and z are vectors containing the state, control and algebraic variables, respectively)

Getting rid of the “loop inside loop” and deriving an efficient way to solve the whole problem

Our new implementation combines both the integration equations and the model equations into a single nonlinear equation set, which ASCEND solves only once for each time step. ASCEND’s compiler combined with its solver can often find surprisingly efficient ways to partition and solve the merged problem, a very nice added benefit. Our goal is to create a package that will minimize the number of times the system has to solve the model equations as these typically take almost all the time for integrating large complicated models.

15.2 THE EQUATIONS WE NEED

We use the ideas in the Lawrence Livermore National Laboratory report ¹ describing LSODE to develop this section. A DAE IVP model has two types of equation sets:

(1) the set that models the physical system at the k-th step (those in the top box in Figure15-1 above)

$$\mathbf{f}(\mathbf{dy}/\mathbf{dx}(\mathbf{k}), \mathbf{y}(\mathbf{k}), \mathbf{z}(\mathbf{k}), \mathbf{u}(\mathbf{k}), \mathbf{x}(\mathbf{k})) = \mathbf{0} \quad (15.1)$$

$$\mathbf{h}(\mathbf{y}(\mathbf{k}), \mathbf{z}(\mathbf{k}), \mathbf{u}(\mathbf{k}), \mathbf{x}(\mathbf{k})) = \mathbf{0} \quad (15.2)$$

$$\mathbf{u}(\mathbf{k}) = \mathbf{u}(\mathbf{x}(\mathbf{k})) \quad (15.3)$$

where \mathbf{y} are the state, \mathbf{z} the algebraic and \mathbf{u} the forcing variable vectors and \mathbf{x} the scalar independent variable, and where \mathbf{f} is a vector of functions equal to the number of states, \mathbf{h} is a vector of algebraic equations equal in number to the algebraic variables, and where one typically specifies the forcing variables as functions only of the independent variable. These the user must provide.

We should note that often one writes equations 1.1 in the form

$$\mathbf{dy}/\mathbf{dx}[\mathbf{k}] = \mathbf{f}^*(\mathbf{y}(\mathbf{k}), \mathbf{z}(\mathbf{k}), \mathbf{u}(\mathbf{k}), \mathbf{x}(\mathbf{k}))$$

where each $\mathbf{dy}/\mathbf{dx}[\mathbf{k}]$ is given on the LHS of its own equation.

(2) the numerical multistep integration equations (those in the bottom box). Multistep integration equations are of the general form

$$\mathbf{y}(\mathbf{k}) = \mathbf{y}(\mathbf{k}-1) + \mathbf{s}(\mathbf{x}[\mathbf{j}]; \mathbf{y}[\mathbf{j}-1], \mathbf{dy}/\mathbf{dx}[\mathbf{j}]; \text{for } \mathbf{j} = \mathbf{k}, \mathbf{k}-1, \dots) \quad (15.4)$$

These integration equations relate the states at the current point to the time derivatives at current and past points and states at past points. They are not related to the model of the physical system but only to the numerical method being used.

Polynomials are in form of Taylor Series expansions around current point

We express the multistep integration equations we shall use here as Taylor series expansions around the current point²:

$$\mathbf{y}(\mathbf{Dx}) = [\mathbf{y}](\mathbf{0}) + [\mathbf{dy}/\mathbf{dx}](\mathbf{0})\mathbf{Dx} + [\mathbf{d}^2\mathbf{y}/\mathbf{dx}^2](\mathbf{0})\mathbf{Dx}^2/2! + \dots \quad (15.5)$$

1. Radhakrishnan, K.R., and A. C. Hindmarsh, Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations, Lawrence Livermore National Laboratory Report UCRL-ID-113855, NASA Ref. Publ. 1327, Lewis Research Center, Cleveland, OH, USA, December (1993).

We also explicitly write their corresponding first derivatives with respect to x :

$$dy/dx(Dx) = [dydx](0) + [d^2ydx^2](0)Dx + [d^3ydx^3](0)Dx^2/2! + ..(15.6)$$

Multistep methods state that these polynomials pass through past data, with the exact data one uses dictating which integration method one is using. Note that the terms $[y](0)$, $[dydx](0)$, $[d^2ydx^2](0)$, etc., are the coefficients in these polynomials. If based on data fitting, their values are estimates for the variables and their derivatives (as implied by the naming we use for them) at the current point.

BDF for stiff equations

For example, a BDF method for integrating stiff ODEs states that the polynomial and its first derivative must satisfy the following equations for a q -th order expansion (i.e., where Equation 15.5 has Δx^q as its highest order term):

- write Equation 15.6 (derivative) for the current point, k .
- write Equation 15.5 (state) for past points $k-1, k-2, \dots, k-q$

“Writing an equation for a point” means to write the equation substituting in the numerical value of Δx corresponding to that point as fixed input on the RHS of the equation and using the numerical value of the state or its derivative as produced by the physical model for that point on the LHS.

Adams-Moulton for nonstiff equations

For the Adams-Moulton method for nonstiff equations, the polynomial and its derivatives must satisfy the following for a q -th order expansion:

- write Equation 15.6 (derivative) for the current point k
- write Equation 15.5 (state) for the previous point, $k-1$
- write Equation 15.6 (derivative) for past points $k-1, k-2, \dots, k-(q-1)$

In both cases, we will have written for each state variable $q+1$ equations in the $q+1$ unknowns $[y](0)$, $[dydx](0)$, $[d^2ydx^2](0)$, ... $[d^qydx^q](0)$, which we can solve as a set of $q+1$ linear equations for each state variable.

2. According to the Fundamental Theorem of Polynomials, all q -th order polynomials passing through the same $q+1$ points are equivalent, no matter how they are written. We choose our polynomial to be in the form of the Taylor Series expansion because of its convenience.

In both cases we have written Equation 15.6 for the current point. We cannot know its LHS value until we have solved for the current point. Thus both methods are implicit - i.e., they require iterative solving even to fit the polynomials.

Looking back at the two boxes in Figure 15-1, the lower integration box will have these $q+1$ equations per state variable in it plus it must have a means to save tabulated numerical values for the states, their derivatives and their value of x at previous points. In principle, it receives, as input, values for $dy/dx(0)$ from the model box.

To complete this box, we include in it one more copy per state variable of Equation 15.5 written at the current point. This last polynomial for each state provides the equation needed to provide the value of the state at the current point, $y(0)$, which we show as the output from this box. The bottom box thus contains $q+2$ polynomial equations in it for each state variable, in terms of the $q+1$ unknown derivatives for each state. The model equations introduce the states $y(0)$ and their derivatives $dy/dx(0)$ for the current point and one equation for each state to compute the latter in terms of the former for it, often in the form

$$dy/dx(0)[i] = f[i](\text{all } y(0), x).$$

So it introduces, in principle, two variables and one equation for each state.

$q+3$ equations in $q+3$ unknowns

To summarize, the integration box introduces $q+2$ equations and $q+1$ new variables per state while the model box introduces 1 equation and 2 new variables. The model is square and, in principle, solvable.

15.3 SOLVING THE CURRENT STEP

Integrating is

- to solve first for the initial point

and then to solve step by step (march) from that point vs x until we reach some stopping condition. We march by

- choosing the integration method,
- determining the step size we should use,
- selecting the order “ q ” to use for that method,
- “predicting” the states at the new current point,

- and finally “correcting” by iteratively solving the model and integration equations for that point.

We shall discuss each of these activities in more detail.

15.3.1 SOLVING FOR THE INITIAL POINT

Solving for the initial point is simply using ASCEND as a solver for a “square” set of nonlinear algebraic equations

Solving for the initial point involves solving the model equations, Equations 15.1 and 15.2. Note we do not insist that one include Equations 15.3 as we may actually want to compute some of the forcing variables at the initial point. If we seek a point where nothing is changing with time, we set all the “fixed” flags for all the derivatives at the initial point to TRUE and set the derivative values to zero. But there is no requirement we limit ourselves to solving for a nontime varying initial point. We really only need to start with these equations, make them into a square set involving an equal number of equations and computed variables and solve using the ASCEND nonlinear equation solver.

As an example, suppose we are filling a tank with water at the rate of a gallon per minute and that the tank has a partially open drain valve but whose fraction open we do not know. We want as the initial point the instant when the tank level is half full but is rising at 1 {cm/s}. We thus set the level (a state variable) to half full and its derivative to 1 {cm/s} and fix them (set their respective “fixed” flags to TRUE). We also fix the flow in and set its value to 1 {gal/min}. The model will have an equation that indicates the flow out through the bottom valve as a function of how far open the valve is (e.g., the valve C_v value) and the level of liquid in the tank. We set the fixed flag for C_v to FALSE so the model computes its value. We finish by setting other fixed flags as appropriate and solve for the initial point. Once solved, we may use the computed value for C_v as a constant in a forcing function that dictates its subsequent value as a function only of time.

15.3.2 CHOOSING THE INTEGRATION METHOD

This package supports two integration methods: BDF and Adams-Moulton (AM). The BDF method is for stiff problems while the AM method is for nonstiff problems. We shall use the definition that says a problem is stiff if the step size one can take to keep the integration computation stable is extremely small compared to the integration interval of interest. Many definitions suggest stiffness is related to the ratio of the largest to the small eigenvalue in the linearized form for two or more ODEs.

The definition of “stiff” that we choose to use here

However, even a single ODE can be stiff using our preferred definition - e.g.,

$$dy/dt = \cos(t) + 10^6*(y-\sin(t)); y(0) = 0$$

is the smooth relatively slow changing function $\sin(t)$, but the second term on the RHS magnifies any numerical error, causing a nonstiff integrator to go unstable unless it takes very small steps.

In other words a problem is stiff if it requires too many steps to solve it stably.

In this package the user selects the method at the start, and it remains unchanged throughout. We, or a programmer skilled in editing the Tcl script for this package, could easily implement a different policy.

15.3.3 DETERMINING THE STEP SIZE WE SHOULD USE

Set stepsize to give prescribed integration error

The last term in the Taylor series expansion for $y[i]$ estimates the error for taking the current step (the primary reason we have selected this form for our polynomials). To determine the allowable step size for each $y[i]$, we solve for δ in the following

$$\frac{d^q y[i]}{dx^q} \cdot \frac{(\delta)^q}{q!} = \text{user prescribed integration error} \quad (15.7)$$

where the numerical value for $\frac{d^q y[i]}{dx^q}$ is that which the converged

integration equations estimated for it (i.e., the coefficient $[d^q y dx^q](0)$) for the current point. The allowable step is then the smallest δ we compute for each of the $y[i]$.

15.3.4 SELECTING THE ORDER “Q” TO USE FOR THAT METHOD

Adjust order of method to allow largest stepsize

This package initially sets the order q to unity.

From then on adjusting the order q is automatic and proceeds as follows.

1. Take q steps with the current order
2. Compute the allowed step for orders $q-1$ (unless q is at its minimum value, 1), q and $q+1$ (unless q is at its maximum value,

- 6). We discuss how to do this computation in a moment.
3. Select as the new current order whichever of these orders allows the largest next step to be taken
4. Return to step 1

Note, we allow q to change by at most plus or minus one whenever we apply this testing.

We carry out step 2 as follows. Use Equation 15.7 to compute the allowed step size for each state variable for the current order, q . The step size allowed is then the smallest of the step sizes one has just computed.

For order $q-1$, the next to last term in the Taylor Series estimates the error; we apply Equation 1.7 to find the step size using this term. Interestingly, the step size allowed that we compute here may be larger than for the higher order polynomial.

A trick to estimate error when increasing order

For order $q+1$, we have no estimate for the appropriate term in the Taylor Series expansion. We can estimate the required derivative as the “divided difference” in the q -th order derivative for the last two points k and $k-1$, namely:

$$\frac{d^{q+1}}{dx^{q+1}}y[i]_k \approx \frac{\frac{d^q}{dx^q}y[i]_k - \frac{d^q}{dx^q}y[i]_{k-1}}{x_k - x_{k-1}} \quad (15.8)$$

To allow us to carry out this computation without added work, we save the past last term in the Taylor Series expansion for each variable and also the last step we took, i.e., $x_k - x_{k-1}$, as we integrate forward.

15.3.5 PREDICTING THE STATES AT THE NEW CURRENT POINT

Predicting is easy after shifting all Taylor Series expansions to the new current point

We can readily predict future behavior for those variables for which we have polynomials fitting past behavior. Our approach is as follows.

1. Determine the order, q , and the step size, d , in the independent variable (previous section) to use next.

2. Shift the independent variable from the “old current point” x to the “new current point” $x+d$. This new current point is to be the point around which we wish to have the new Taylor Series expansions for all the state variables.
3. Shift all the “old Taylor Series expansions” around the old current point so they represent a “new Taylor Series expansion” around the new current point. Shifting will involve computing new values for each of the Taylor Series coefficients - i.e., $[y](0)$, $[dydx](0)$, $[d^2ydx^2](0)$, and so forth. These new values are an easily computed function of the size of the shift, δ , and the old coefficients. The shifted polynomial still fits the data we knew about for the previous point but not for the new point - i.e., it is a polynomial based on earlier behavior, but that is why we only are predicting when using it.

Assuming we have completed the shift. Then, looking at Equation 15.5

$$y(\Delta x) = [y](0) + [dydx](0)\Delta x + [d^2ydx^2](0)\Delta x^2/2! + \dots,$$

We note that at the new current point, Δx is zero. All terms except the first are multiplied by powers of Δx and are, therefore, zero at this point. Thus, the zeroth Taylor Series coefficient, $[y](0)$, for each variable predicts that variable at the current point.

To repeat - prediction involves shifting the coefficients for the Taylor series polynomial for each variable so the expansion is around the new current point, and then the zeroth Taylor Series coefficient is the prediction. Correction, which we discuss in the next subsection, is to recompute all the Taylor Series coefficients so the new polynomial fits the appropriate past data and the model equations at the current point. If past behavior is a good prediction, this recomputation should not change the coefficients very much.

15.3.5.1 SHIFTING COEFFICIENTS IN THE TAYLOR SERIES POLYNOMIALS

You can skip reading this subsection unless you wish to see the details of the shifting. The ideas are straight forward, even if the algebra seems to get a bit complicated.

We carry out the shifting in Steps 2 and 3 above as follows.

First we note that

$$Dx = x - x_0 = x - x_{0,new} + x_{0,new} - x_0 = Dx_{new} + d$$

where we have expanded the current polynomials around x_0 . To shift we need to substitute Dx_{new} for Dx in Equation 15.5 and

collect all the “constant” terms as the new $[y](0)$, those first order in $\mathbf{D}x_{new}$ as the new $[dydx](0)$, etc.

Substituting for $\mathbf{D}x$ in Equation 15.5, we get

$$\begin{aligned} y(\mathbf{D}x_{new}) &= [y](0) + [dydx](0)(\mathbf{D}x_{new} + \mathbf{d}) \\ &\quad + [d^2 y dx^2](0)(\mathbf{D}x_{new} + \mathbf{d})^2 / 2! + \dots, \\ &= \{ [y](0) + [dydx](0) * \mathbf{d} + [d^2 y dx^2](0) * \mathbf{d}^2 / 2! + \dots \} \\ &\quad + \{ [dydx](0) + [d^2 y dx^2](0) * 2\mathbf{d} + \dots \} * \mathbf{D}x_{new} \\ &\quad + \dots \end{aligned}$$

The shifted polynomial coefficients are in the curly braces, $\{ \}$, above. Collecting the constant terms, we get:

$$y(0)_{new} = [y](0) + [dydx](0) * \mathbf{d} + [d^2 y dx^2](0) * \mathbf{d}^2 / 2! + \dots$$

Collecting those first order in $\mathbf{D}x_{new}$, we get

$$[dydx](0)_{new} = [dydx](0) + [d^2 y dx^2](0) * 2\mathbf{d} + \dots$$

In a similar manner we can find the higher derivatives. The terms come from expanding $(\mathbf{D}x_{new} + \mathbf{d})$ to higher and higher powers, which we can do quickly using Pascals triangle to develop the coefficients. Such a triangle for expanding to powers up to and including five is:

\col	(0)	(1)	(2)	(3)	(4)	(5)
row (0):	1	1	1	1	1	1
row (1):	1	2	3	4	5	
row (2):	1	3	6	10		
row (3):	1	4	10			
row (4):	1	5				
row (5):	1					

Using this triangle, we readily write down the expansion for $(\mathbf{D}x_{new} + \mathbf{d})^5$ as

$$\mathbf{D}x_{new}^5 + 5\mathbf{D}x_{new}^4 \mathbf{d} + 10\mathbf{D}x_{new}^3 \mathbf{d}^2 + 10\mathbf{D}x_{new}^2 \mathbf{d}^3 + 5\mathbf{D}x_{new} \mathbf{d}^4 + \mathbf{d}^5$$

The coefficients (1, 5, 10, 10, 5, 1) are the rightmost entries in each row. To form a Pascals triangle, we start each row and column with a 1. We form each subsequent number (starting with the coefficient in row 1, column 1) by adding the numbers just above and just to the left. So the 10 in row 2, column 3 is formed by adding the 4 above to the 6 to the right. Let us denote each such coefficient as PT[row number, column number]; then PT[2,3] = 10. The general shifting equation is then:

$$\left[\frac{d^k y}{dx^k} \right]_{\text{shifted}} = \left[\frac{d^k y}{dx^k} \right] + \sum_{j=1}^{q-k} PT[j, k] \cdot \left[\frac{d^{k+j} y}{dx^{k+j}} \right] \cdot \frac{k!}{j!} \cdot \delta^{j-1} \quad (15.9)$$

15.3.6 CORRECTING BY ITERATIVELY SOLVING THE MODEL AND INTEGRATION EQUATIONS FOR THAT POINT

Solving all the equations for current point is the correction step

Once we have made all the above decisions and preparatory computations, we pass the model for the current point - with the appropriate fixed flags set to TRUE - to the ASCEND solver, which then solves it (iteratively, of course). Solving is the correction step. The model we are solving comprises equations 15.1 to 15.4 - i.e., the equations describing the process dynamics together with the numerical integration equations. In other words, we are solving all the equations in both the inner and outer loops in Figure15-1 simultaneously.

The unknowns are for the current point ($\Delta x=0$) are

- the states y ,
- their first derivatives dy/dx ,
- the algebraic variables z ,
- the control variables u ,
- and finally all the polynomial coefficients ($[y](0)$, $[dydx](0)$, $[d^2ydx^2](0)$, $[d^3ydx^3](0)$, etc.) in Equation15.4 (written in the forms for Equations 15.5 and15.6).

The variables we fix for this computation are

- the independent variable x and
- the past values for the states and their derivatives.

15.4 SOME USEFUL “FINE POINTS”

The previous section outlines the issues related to solving initial value problems by combining the model and integration equations into a single set of equations one will then solve at each time step using ASCEND. We now will discuss three added issues that can be important for solving initial value problems: (1) predicting some of the algebraic variable values at the current time step, (2) defining and using stopping conditions, and (3) detecting index problems.

15.4.1 PREDICTING SOME OF THE ALGEBRAIC VARIABLES

By slightly modifying the ideas we developed above for predicting state variables, we can predict values at the new current point for any of the algebraic variables, should that prove numerically useful. In many problems, the equation structure can allow one to compute many of the problem variables in terms of only a few of them. Thus, if we could predict those few, we could use a method to compute directly and without iteration good guesses for the remaining ones in terms of them, significantly improving our chances for converging quickly and sometimes for converging at all.

The idea is simple: we can fit a polynomial to the past values of an algebraic variable vs. the independent variable. We can then use that polynomial to predict the value for the algebraic variable at the new current point. As we are already writing polynomial equations for the states, this added capability does come at little added effort. The equations we write can only be of the form of Equation 15.5 as we do not have model equations providing derivatives for algebraic variables. This feature is one we have included in this package. To fit a polynomial of order q to algebraic variables, we write the polynomial at the current point (must match the value the model computes) and q past values.

15.4.2 DEFINING AND USING STOPPING CONDITIONS

We integrate our model forward until any one of a set of possible stopping conditions occurs. We identify a stopping condition as the point at which there is the first change of sign of a variable we associate with that stopping condition. So, if we wish to stop if the temperature (T) in a flash unit rises above the temperature at which a mixture in it will boil (T_{dewPt}), we can define the variable $T_{\text{stop}} = T_{\text{dewPt}} - T$. When T_{stop} switches from being positive to negative, we will have hit our stopping condition.

A way to stop at the precise point where the stopping condition changes sign is actually pretty straight forward, conceptually. We alter the model we solve by (1) fixing the value for the variable associated with the stopping condition to be precisely zero and (2) freeing up the value of the independent variable so the model computes exactly when that stopping condition occurs. We do precisely this trading of degrees of freedom in this package.

We may have a number of different stopping conditions, the first of which becoming true causes us to stop our integration. We had in our

opening example of heating a tank filling with liquid that we wanted to stop after ten minutes ($\text{time_stop} = 10 \{\text{minute}\} - x$), or when we have just filled the tank ($\text{lev_stop} = \text{lev_full} - \text{lev}$) or when the liquid first starts to boil ($\text{T_stop} = \text{T_dewPt} - T$), whichever comes first. We monitor each of the stopping variables for a change in sign. We stop when any one of them does so.

The hard part is deciding when to stop if two or more of the conditions change sign in the same time step. We can decide which happened first as follows. We solve the model for the next time step as if no stopping condition will occur in that interval. We then check the stopping conditions to see if one or more of them has had a sign change. If none has, we continue to the next time step. Otherwise, we pick one whose sign switched and set its fixed flag to TRUE and its value to zero. We release the value for the independent variable by setting its fixed flag to FALSE and resolve the model. The computed value for the independent variable is its value at which the stopping condition is precisely zero. We again check if any of the other stopping condition has become true within this reduced step. If so, it has become true before the one we previously selected. It should be our stopping condition. So we fix this variable at zero (and release the previous one) and resolve, getting an even shorter step. We check again for any of the other stopping conditions, continuing until no new stopping condition remains true.

We have limited the variables associated with stopping conditions to being among any of the variables we predict - state or predicted algebraic - only. We can, if we wish, then solve the prediction polynomials for each of them for the value of the independent variable at which the variable value is zero, allowing us to get a good estimate for its final value and to decide which stopping condition is the most likely to have occurred first if several undergo a sign change in the same step. This would be useful if solving each time step were to take a long time - e.g., several minutes to hours.

15.4.3 DETECTING INDEX PROBLEMS

An index problem occurs when solving DAE models because of the manner in which one has written the model equations and/or set up the degrees of freedom for it and not because the model and its underlying physical system does not have a solution. Having an index problem causes one to compute the errors while integrating incorrectly. Essentially it comes about because one has written the model in a way that forces it to use one or more of the integration equations "backwards" as differentiators rather than as integrators. As one has assumed these equations to be used to integrate and based the error

estimation on that assumption, one will typically get much larger errors than expected and “kill” the integration - without the user being any the wiser. The answers could look perfectly reasonable and in fact be garbage instead.

Fortunately, one can detect if one has an index problem in a fairly straight forward manner IF one can detect singularity of the model equations when solved with the degrees of freedom set in a particular way. This ASCEND can easily do as it involves setting the degrees of freedom for the model and using two singularity testing tools in the ASCEND Solver tool set - one to detect structural singularity and the other to detect numerical singularity. Structural singularity is a guaranteed test - i.e., if and only if structural singularity is there, the test detects it. Structural singularity guarantees numerical singularity, so, if one has structural singularity in the testing, one has an index problem. Numerical singularity testing is a “fine” art and is never conclusive. It is just that some tests can be better than others. Ours is based on small pivots and is an “okay” form of test, but it could miss singularities that are really there because of ill-conditioning and that do not show themselves with small pivots.

If one attempts to solve the model equations - i.e., those in the upper box in Figure15-1 on page2 - with the states as inputs and the derivatives and algebraic variables as computed outputs and that computation is singular, one has an index problem. One should take steps to eliminate it before proceeding. Eliminating an index problem can be very complicated, or it could be as simple as fixing a currently computed variable while releasing another that is currently being computed - i.e., exchanging the roles of a fixed variable and a computed variable.

Detection is, therefore, to have the user provide a method that sets up this computation for just the user model without the integration equations and to pass this part of the model to the solver, which will complain before solving if there is a structural singularity problem. If the structural test passes, one can then actually solve these equations and test for numerical singularity. It might be easiest to do this testing by stopping the integration part way along (set the integration to stop after one second, for example), running the degree of freedom setting method just described, and testing with the numerical singularity testing tool located in the ASCEND Solver tool set. At that point, the system will have converged the equations so the test will be much better than running it where the equations are not converged.

15.5 MANAGING THE INTEGRATION PROCESS IN

ASCEND

The integration process involves solving the initial point and then stepping the models from the initial x until a stopping condition is true. As one steps, one is developing the trajectories for all the variables vs. x . Viewing the results of integration is typically to plot values for a selection of the variables vs. x .

We have already discussed how one can solve for the initial point in Section 15.3.1 on page 6. The remainder of the previous section then discusses how one can set up and solve a single step, which, when integrating, we do repeatedly until we reach a stopping condition. Managing this process is the subject of this section.

15.5.1 THE STRUCTURE FOR THE ASCEND MODEL

An initial value problem is one in which we know the initial conditions for a model and can then compute one step at a time as we march forward vs. the independent variable x . Thus we need only have the equations for the current point available at any one time while solving. In contrast, a two point boundary value problem is one where we know some of the initial conditions - but not enough to allow us to solve the model there - and some of the final conditions. We are required to write all the equations for all the time steps and solve them all simultaneously for such a problem. If we need a thousand time steps, the model we need to pose and solve is 1000 times larger than we need to solve an initial value problem.

We propose the following structure for an ASCEND model for solving initial value problems. We shall require the user to supply a model that supplies equations 15.1 to 15.3 - i.e., the physical model that relates the states to their derivatives. This model will typically involve algebraic and forcing variables z and u , along with equations to compute them. We need to set up and solve these equations for the current time step only. Once we solve the current step, we save the results as historical data, step the independent variable forward to a new current point, and apply the same equations to solve for this new point.

We can use a script in ASCEND to carry out the solving process. The script will proceed as follows.

1. Load and compile the ASCEND model the user supplies (Equations 15.1 to 15.3). The user model has parts that are the

integration system models (Equations 15.4); thus both sets of equations are in the final model for the current point.

2. Use user supplied methods to establish the correct degrees of freedom (Section 15.3.1 on page 6) and values for variables at the initial point and then solve the model at the initial point.
3. Use a user supplied method to establish the correct degrees of freedom and any added values for solving a current point when stepping (Section 15.3.6 on page 11).
4. Set up to use the integration method (Adams-Moulton for nonstiff and BDF for stiff - Section 15.3.2 on page 6) that the user has chosen.
5. Set the polynomial order (initially to unity) and the step size for the independent variable x (initially to a value that the user prescribes).
6. Prepare to compute the next point as follows -
 - move previous points backward in a data table set up to capture them (only q past points are ever needed so points older than this drop out of the table),
 - increment x and
 - predict values for all states at the new current point by shifting the polynomial coefficients (Section 15.3.5 on page 8).
7. Solve model and integration equations at current point (Section 15.3.6 on page 11).
8. Stop if a stopping condition met (Section 15.4.2 on page 12), else determine polynomial order and step size for next step (Sections 15.3.3 and 15.3.4).
9. Repeat from Step 5.

We have created a Tcl program and several methods within the system supplied models to implement these steps. A system supplied script for this capability calls the Tcl program.

15.6 EXAMPLE 1: FILLING A TANK WITH WATER

15.7 EXAMPLE 2: CHANGING THE FEED TEMPERATURE INTO A PRESSURE CONTROLLED, RIGOROUS FLASH UNIT