Differential Equation Models and Their Simulators


In discrete time modeling we had a state transition function which gave us the information of the state at the next time instant given the current state and input. In the classical modeling approach of differential equations, the state transition relation is quite different. For differential equation models we do not specify a next state directly but use a derivative function to specify the rate of change of the state variables. At any particular time instant on the time axis, given a state and an input value, we only know the rate of change of the state. From this information, the state at any point in the future has to be computed.

To discuss this issue, let us consider the most elementary continuous system – the simple integrator (0). The integrator has one input variable $x$ and one output variable $y$. One can imagine it as a reservoir with infinite capacity. Whatever is put into the reservoir is accumulated – but a negative input value means a withdrawal. The output of the reservoir is its current contents. When we want to express this in equation form we need a variable to represent the current contents. This is our state variable $q$. The current input $x$ represents the rate of current change of the contents which we express by equation

$$\frac{dq(t)}{dt} = x(t)$$

and the output $y$ is equal to the current state

$$y(t) = q(t).$$
Usually continuous systems are expressed by using several state variables. Derivatives are then functions of some, or all, the state variables. Let \( q_1, q_2, \ldots, q_n \) be the state variables and \( x_1, x_2, \ldots, x_m \) be the input variables, then a continuous model is formed by a set of first-order differential equations:

\[
\frac{d}{dt} q_1(t) = f_1(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t)) \\
\frac{d}{dt} q_2(t) = f_2(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t)) \\
\cdots \\
\frac{d}{dt} q_n(t) = f_n(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t))
\]

Note that the derivatives of the state variables \( q_i \) are computed respectively, by functions \( f_i \) which have the state and input vectors as arguments. This can be shown in diagrammatic form as in Figure 10. The state and input vector are input to the rate of change functions \( f_i \). Those provide as output the derivatives \( dq_i/dt \) of the state variables \( q_i \) which are forwarded to integrator blocks. The outputs of the integrator blocks are the state variables \( q_i \).

![Figure 11: Structure of differential equation specified systems](image)

### 3.1.1 Continuous system simulation

The diagram above reveals the fundamental problem that occurs when a continuous system is simulated on a digital computer. In the diagram we see that given a state vector \( q \) and an input vector \( x \) for a particular time instant \( t_i \) we only obtain the derivatives \( dq_i/dt \). But how do we obtain the dynamic behavior of the system over time? In other words,
how do we obtain the state values after this time? This problem is depicted in 0. In digital simulation, there is necessarily a next computation instant $t_{i+1}$ and a nonzero interval $[t_i, t_{i+1}]$. The model is supposed to be operating in continuous time over this interval, and the input, state and output variables change continuously during this period. The computer program has available only the values at $t_i$ and from those it must estimate the values at $t_{i+1}$ without knowledge of what is happening in the gaps between $t_i$ and $t_{i+1}$. This means it must do the calculation without having computed the input, state, and output trajectories associated with the interval $(t_i, t_{i+1})$.

Schemes for solving this problem are generally known as numerical integration methods. A whole literature deals with design and analysis of such methods [Press et al 92, Burden, Faires 89]. Here we try only to provide some insight into the operation, accuracy, and complexity of these methods from a broader systems theoretic perspective.

Let us study the approach by considering the simplest integration method, generally known as the Euler or rectangular method. The idea underlying the Euler method is that for a perfect integrator

$$\frac{dq(t)}{dt} = \lim_{h \to 0} \frac{q(t+h) - q(t)}{h}$$

thus for small enough $h$, we should be able to use the approximation

$$q(t + h) = q(t) + h \cdot \frac{dq(t)}{dt}.$$  

Although straightforward to apply, Euler integration has some drawbacks. On one hand, to obtain accurate results the step size $h$ has to be sufficiently small. But the smaller the step size the more iterations are needed and hence the greater the computation time for a given length of run. On the other hand the step size also cannot be decreased indefinitely, since the finite word size of digital computers introduces round-off and truncation errors. Therefore, a host of different integration method have been developed which often show much better speed/accuracy tradeoffs than the simple Euler method. However, these methods introduce stability problems as we discuss in a moment.

**Exercise:** Show that the number of iterations varies inversely to the step size in Euler integration. How does reducing the step size an order of magnitude impact the number of iterations?
The basic idea of numerical integration is easily stated – an integration method employs estimated past and/or future values of states, inputs and derivatives in an effort to better estimate a value for the present time \( (t_0) \). Thus to compute a state value \( q(t_i) \) for a present time instance \( t_i \), may involve computed values of states, inputs and derivatives at prior computation times \( t_{i-1}, t_{i-2}, \ldots \) and/or predicted values for the current time \( t_i \) and subsequent time instants \( t_{i+1}, t_{i+2}, \ldots \)

\[
q(t_i) = \text{integration_method}(..., q(t_{i-2}), f(q(t_{i-2}), x(t_{i-2})), q(t_{i-1}), f(q(t_{i-1}), x(t_{i-1})), \, q'(t_i), \, f(q'(t_i), x'(t_i)), \, q'(t_{i+1}), f(q'(t_{i+1}), x'(t_{i+1})), \, ...) 
\]

where \( q' \) and \( x' \) are the state and input estimates.

Notice that the values of the states and the derivatives are mutually interdependent – the integrator itself causes a dependence of the states on the derivatives and through the derivative functions \( f \) the derivatives are dependent on the states. This situation sets up an inherent difficulty which must be faced by every approximation method – the propagation of errors.

![Figure 13: Computing state values at time \( t_i \) based on estimated values at time instants prior and past time \( t_i \)](image)

We now describe briefly the principles of some frequently employed integration methods. We can distinguish methods according to whether or not they use estimated future values of variables to compute the present values. We call a method causal if it only employs values at prior computation instants. A method is called non-causal if in addition to prior values it also employs estimated values at time instants at and after the present time. The order of a method is the number of pairs of derivative/state values it employs to compute the value of the state at time \( t_i \). For example, a method employing the values of \( q \) and its derivative at times \( t_{i-2}, t_{i-1}, t_i, \) and \( t_{i+1} \) is of order 4.

**Causal Methods**

A causal method of order \( m \) is often a linear combination of the state and derivative values at time instants \( t_{i-m} \) to \( t_{i-1} \) with coefficients chosen to minimize the error from the computed estimate to the real value. Usually the basis of the derivation of the coefficients is the assumption that the state trajectories and the output trajectories are smooth enough to be approximated by the \( d \)th order polynomial. Of course, often it is not known beforehand whether the requisite smoothness indeed obtains.
Example.
The Adams method is a simple causal method of order 2 and can be described by
\[ q(t_i) = q(t_{i-1}) + h \left( 3f(q(t_{i-1}), x(t_{i-1})) - f(q(t_{i-2}), x(t_{i-2})) \right). \]

A general problem in causal integration methods is the startup, namely, the determination of the state, derivative, and output values for the times \( t_{1-m} \) to \( t_1 \) at the initial simulation time \( t_1 \). This is generally referred to as the startup problem. A solution to this problem is to use causal methods of lower orders in the startup phase. That is, in the first cycle, where only the initial state is available, a method of order 1 is used and subsequently the order of the method is increased to its final order \( m \). Another, more reliable solution is to change the method completely for the startup phase. For example, to use a non-causal method (see below) in the startup phase when past values are not available but future values can be estimated.

**Non-Causal Methods**

As we have indicated, the non-causal integration methods make use of “future“ values of states, derivatives and inputs. Clearly, the only way of obtaining "future" values is to run the simulation out past the present model time, calculate and store the needed values, and use these data to estimate the present values. Thus the values at the present time may be computed twice – tentatively, during the initial “predictor” phase when the future is simulated, and once again during the “corrector“ phase when the final value is computed. A simulator program with a non-causal method, therefore, has to work in several phases. At first there may be several predictor phases and finally there is usually one corrector phase.

**Example.**

A simple predictor-corrector method generally known as the Heun-method is as follows
\[ q'(t_i) = q(t_{i-1}) + h f(q(t_{i-1}), x(t_{i-1})) \]
\[ q(t_i) = q(t_{i-1}) + \frac{h}{2} \left( f(q'(t_i), x'(t_i)) + f(q(t_{i-1}), x(t_{i-1})) \right) \]
where \( q'(t_i) \) is the predicted value and \( q(t_i) \) is the final corrected value.

In some methods, called variable stepsize methods, the difference between two computed values is compared with a criterion level; if the estimated error is too large, the step size is decreased and the predictor – corrector cycle repeated until a step size is found for which the difference is below the criterion level. Of course, if the state trajectory is not sufficiently smooth, no finite step size may be found for which the divergence is sufficiently small, and the simulation will drive to an expensive halt.

Many non-causal methods, among them the most popular Runge-Kutta methods, do not use past values but only use the current state and input value to make the predictions of future values. For those methods, the startup phase is not a problem. Such methods therefore are also good alternatives to employ in the startup phase of causal methods.

**To Euler or Not: That is the Question**

Error arises from two sources, that introduced in each step by the approximation method and that accumulated through the effect of prior error propagating through the system. In case of integration methods, the first source of errors exists even if the values of state and derivative are correct. The second source of error is due to the mutual dependence of states and derivatives just mentioned. An error in state is transmitted to the derivative functions, where it may further affect the state values. An integration method may tend to
amplify or dampen the effect of error feedback. All other things being equal, a so-called *stable* method (one that dampens error propagation) is preferable to one that amplifies the error propagation. But the error propagation does not only depend on the method, it also depends on the nature of the model – whether it tends to amplify or dampen deviations (e.g., whether trajectories emerging from initial states close together tend to remain close together). From this we can see that the choice of integration method, or for that matter any simulation algorithm, for a given model is not an easy one.

We have suggested that multi-point methods can be faster than direct Euler integration while preserving accuracy. However, these methods also have their drawbacks. They introduce stability and start up problems that Euler doesn’t suffer from. They make assumptions about the analytic nature of the trajectories that can easily be violated in hybrid models with state events (see Chapter 9). Also, with modern high speed computers, the use of Euler and other direct integration methods is becoming more practical. Fortunately, modern environments and packages for continuous simulation shield the modeler from many of these considerations. Indeed, in principle, users should be able to work with models independently of the underlying integration method. Unfortunately, however, today’s environments are not perfect and the user has to be on the look out for signs that the simulator is not faithfully generating the model trajectories. For example, puzzling results may be due to the integration becoming unstable rather than an interesting model behavior. Or more perniciously, innocent looking simulated trajectories may hide unsuspected integration errors. Table 1 summarizes the pros and cons of integration methods.

<table>
<thead>
<tr>
<th>Integration Method</th>
<th>Efficiency</th>
<th>Start-up Problems</th>
<th>Stability Problems</th>
<th>Robustness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euler</td>
<td>low</td>
<td>none</td>
<td>not for sufficiently small step sizes</td>
<td>high</td>
</tr>
<tr>
<td>Causal methods</td>
<td>high</td>
<td>yes</td>
<td>yes</td>
<td>low</td>
</tr>
<tr>
<td>Non-Causal methods</td>
<td>high</td>
<td>no, if only future values used</td>
<td>yes</td>
<td>low</td>
</tr>
</tbody>
</table>

Table 1 Summary of Integration Method Speed-Accuracy Tradeoffs

The problems of error introduction and error propagation are endemic to simulator and model construction throughout the whole M&S enterprise. We will take up these issues later in Chapters 13 and 14.

3.1.2 Feedback in continuous systems

After acquiring a basic understanding of differential equation models and how they are simulated on digital computers, let us now try to get some insight into the nature of continuous system behavior. By considering several elementary systems we will demonstrate that the reason for the emergence of quite complex behavior is *feedback*. In a system model embodying feedback loops state variables are fed back to influence their own rates of change. Feeding back a state variable to its own derivative can be direct or
also involve several other state variables. 0 shows a feedback loop where a state variable \( q_1 \) influences the derivative of \( q_2 \) which again influences \( q_3 \) and so on. Finally a state variable \( q_n \) is used to define the derivative of \( q_1 \) closing the feedback loop.

![Feedback loop](image)

**Figure 14 Feedback loop**

Qualitative analysis of the feedback loops in a system can give insights into its possible behaviors. Most important is whether a feedback loop is positive or negative. When traversing the feedback loop we can count the signs of the direct influences between the state variables. When the sign of a function \( f_i \) is positive, we speak of a positive influence from \( q_{i-1} \) to \( q_i \), which means that a positive value of \( q_{i-1} \) will make \( q_i \) increase. Conversely, when the sign of a function \( f_i \) is negative, a positive value of \( q_{i-1} \) will cause a decrease of \( q_i \). A positive feedback loop is one with an even number of negative influences. A negative feedback loop is one with an odd number of negative influences.

How do they differ? Let us consider as an example a positive feedback loop with zero negative influences and a negative feedback loop with one negative influence. In the positive feedback loop a greater value of a state variable will cause all influenced variables in the feedback loop to increase. In turn, this will cause its own derivative to increase and the variable will get even greater. This feedback loop causes the variable grow indefinitely. In some contexts, this is called unstable behavior and is undesirable. However, in other contexts, such as biological and economic growth, this is very desirable (at least for some part of the trajectory). For negative feedback loops, however, the negative influence will tend to stabilize a system. Growth of the state variable will tend to be turned into negative influence on its derivative which would cause the state variable to decrease (we say “tend”, because, the situation is a actually more complex, see [Levins, Puccia 86]).

**Exercise:** Argue why a feedback loop with an even number of negative influences works like one with no negative influence and one with an odd number of negative influences is the same as one with one negative influence.

### 3.1.3 Elementary Linear Systems

After this general discussion of feedback in continuous systems, let us introduce some well known basic continuous systems and study their behavior. These often appear as elementary substructures in models of complex real phenomenon independently of the application domain at hand (the feedback takes different forms in the different structures.) As we will see, simple continuous systems can show quite interesting behavior, despite their simple-looking structure. The reason for this is feedback.

First of all let us discuss a system with one state variable with a direct feedback loop to itself. The derivative of the state variable is computed by the state variable multiplied with a linear factor \( c \). When the factor \( c \) is positive, we have a positive feedback. In this case, the system will grow exponentially and we speak of an exponential growth.
Conversely, when the factor \( c \) is negative, we have a negative feedback and the variable will shrink approaching zero exponentially. We have a exponential decay. When the factor is zero, the state remains constant. 0 shows the system structure and several different behaviors for different values for factor \( c \). The larger \( c \) is in magnitude, the steeper the curve whether increasing or decreasing.

\[ c \]

\[ q \]

As illustrated in 0, when we apply an input to an exponential decay, we have the so-called exponential delay of first order. This type of continuous system plays an important role in many real world phenomenon. In abstract terms, this is the right type of model to use for system with a driving force and a linear damping. The input to the system represents the driving force and the feedback through negative factor \( c \) models the damping. These two are added defining the derivative for the state variable. The characteristic of the system is that the system reaches an equilibrium state when the damping through the feedback \( -c*q \) equals the driving input. 0 shows this for a constant input force \( x_1 \) and different damping factors \( c \). The greater the damping factor, the smaller the equilibrium state.

There are many real world phenomenon which can be modeled by an exponential delay of first order. Examples range from heating systems where the input is the heat supply and the damping represents the heat losses to the environment, to mechanical systems where the input is the force supplied and the damping represents the friction which should be counteracted, to input of pollution into a natural system and their reduction through an absorption process.

Figure 15 Exponential growth and decay
The examples above are systems with one state variable with a direct feedback loop to itself. Let us now consider a system in two variables. The linear (or harmonic) oscillator of second order is the most well-known example where the effect of feedback can easily be studied.

A mechanical oscillator can serve as a reference system to discuss the equations. In the mechanical oscillator we have a mass $m$ which is connected to some fixed device through a spring and a damper. Both have linear characteristics. The spring is defined by a spring constant $k$ and the damper by factor $d$. A force $F(t)$ is applied to the mass over some time period.

The system is modeled by two state variables, viz. the current position $x$ of the mass relative to the resting position, and the current velocity $v$ of the mass. These variables influence each other. By definition, $v$ is the derivative of $x$. The feedback from $x$ to $v$ is through the force of the spring, namely, depending on the current deflection $x$ of the mass, the spring will counteract and influence the velocity $v$ by $-k \cdot x$. Also, damping exerts a direct negative feedback from the velocity to itself ($d \cdot v$). The greater the velocity, the greater the damping gets.
Most interesting for the behavior of the system is the negative feedback loop from the position $x$ to the velocity $v$ due to the spring force. Rather than cause growth or decay, this feedback actually causes the system to oscillate. Let us analyze the situation in little more detail. In $0$, we start with the mass at position 1 with velocity being 0. Now we let the mass loose. The effect will be that the spring force will accelerate the mass towards its resting position. When the mass has reached the zero position, the mass has acquired its maximum velocity and the kinetic energy will move the mass into the opposite direction. On the other side, however, the spring will act in the opposite direction, decelerating the mass until it comes to a rest and finally accelerating the mass back to its resting position again. Therefore, the system oscillates. If the damping factor is strong, the amplitude of the oscillation will rapidly decrease bringing the system to a halt. In some situations, the “damping” can be positive. In this case, it adds, rather than removes, energy from the system.

**Exercise:** Show that a system with positive damping oscillates with exponentially increasing amplitude (until something stops it – such as exploding into another system!).

If we remove damping altogether, we get a pure oscillator as shown in $0$. Here the spring or other system such as an electronic circuit with no resistance keeps “ringing” forever. In the figure, each integrator has the same gain, $\omega$, (although the signs differ) and the oscillation has frequency $\omega$ (with period $2\pi/\omega$). It is described by the familiar $\sin \omega t$ and $\cos \omega t$ curves shown. Such a system is called *neutrally* stable since it is neither
damped nor undamped. If we add a small value to the state of an integrator, it will incorporate this change into the magnitude of its oscillation. We will see later (Chapter 16) that this is really the worst environment in which a simulator or approximation procedure has to operate since every error accumulates in both an absolute and relative sense.

**Exercise.** Relate $\omega$ to $k$ in $0$ and thereby provide an expression for the frequency of oscillation in the damped spring shown there.

**Oscillators**

![Oscillators](image)

Figure 19 Undamped Second Order Linear Oscillator: (a) time trajectories, (b) phase space

Now that we have also gained a general understanding of continuous modeling and simulation, let us come back to the comparison of the linear oscillator above and the discrete time oscillator presented earlier. Although the similarities of the two models are obvious and they also show comparable behavior, they are also very different. By their differences we also obtain some insight into discrete time and continuous modeling in general.

The biggest correspondence between the two models is that both have a negative feedback which is the reason why they oscillate. The main difference, however, comes from the fact that in discrete time models, the inputs to the delays define new state values while in the continuous domain the inputs represent rate of change values for the state variables. Let us consider the gains in the continuous and the discrete oscillators. In the discrete oscillator, the gain determines the amplification or decay of the value of one delay as it is fed to the other. In the continuous oscillator, the gain amplifies the rate of change of the influenced variable and thereby defines how fast the change of the state occurs. It determines the frequency of the oscillation. How is the frequency defined in the discrete domain? Actually by two things, namely by the clock rate and second by the number of delays in the feedback loop. Recall that in the discrete oscillator, we had two delays, and the sign of the values changed every two clock ticks. If we introduce a further delay, it will take another tick that a change in sign due to the negative feedback propagates through all the delays. An oscillation frequency of three clock ticks results. Thus the mechanism underlying the oscillation is quite different in the discrete and continuous cases.
3.1.4 Non-Linear Oscillators: Limit Cycles and Chaotic Behaviors

The systems we have considered so far were all linear. Now we discuss a non-linear system – the so-called Van der Pol system which comes into being by adding a non-linear feedback to the linear oscillator. Instead of the linear damping factor $-d$ a non-linear feedback is introduced which is dependent on the value of $x$, namely the feedback is equal to $(1 - x^2)\cdot v$. 0 a) shows the model structure.

Through this non-linear modification, the system acquires some quite astonishing properties. Independent of its initial state values, the system quickly comes into a stable oscillation. The reason for this is that for small values of $x$, the factor $(1 - x^2)$ is positive, thus leading to an amplification of the value $v$, whereas for large values of $x$, the factor $(1 - x^2)$ rapidly gets negative which leads to fast damping of $v$. The effect is that for small values of $x$, the feedback drives the values out of the region around zero but as soon as the value is above 1 or $-1$, the factor gets negative and drives the value of $x$ back. The result is a stable oscillation as shown in 0. Because the trajectory of the system always returns to a closed loop in state space, the latter is called a limit cycle.

Exercise: Compare cyclic behavior of the Van der Pol system with that of the 2nd order linear oscillator.
Lotka-Volterra predatory-prey systems form an interesting class of non-linear models that exhibit oscillatory behavior — although not as limit cycles. As shown in 0 b), the integrator outputs represent prey and predator populations. In this case, the feedback parameters of each integrator is a non linear function of the other output.

Exercise: Find the equilibrium point of the Lotka-Volterra model and investigate the oscillations around this equilibrium. Where do the maximum and minimum populations occur? Show that small oscillations around the equilibrium are approximated by the 2nd order linear oscillator.

While non-linear systems can show remarkable stability properties, they can also exhibit highly unstable behaviors that appear chaotic. Indeed, the term chaos is used by mathematicians for the patterns underlying such “strange” behaviors [Alligood et al. 96]. Chaotic behavior can come into existence by simple non-linear feedback. The so-called Rössler attractor is a modified linear oscillator in three variables (0). Its typical characteristic is that, independently of the initial condition, the system will neither come to rest nor to some periodic behavior (0). The behavior appears to be unpredictable to an observer, but actually a pattern may be discerned. In the state space one can observe that the system actually oscillates around the zero point, but it never takes exactly the same paths. The system is linear except for the coupling from x to z. In this coupling, the expression \((x - c)\) causes a change in sign and this is the reason that the system exhibits non-repetitive fluctuations. These look like random noise, showing the close connection between complex deterministic and probabilistic behaviors (see Chapter 10).
\[ f_i (x, z) = b + (x - c)z \]

**Figure 22** Rössler system

*Figure 23 Chaotic behavior of Rössler system: (a) state plane \( v \) and \( z \) to \( x \), (b) time trajectories*

**Exercise**: Compare cyclic behaviors of the Rössler, Van der Pol and 2\(^{nd}\) order linear oscillator.

**Exercise**: As with randomness, chaotic behavior can have useful “statistical” properties. For example, the manufacturer of a mobile swimming pool vacuum cleaner certifies that it will cover every inch of the surface every 12 hours. Yet the cleaner never exactly repeats its trajectory through the pool. How might a model exhibiting chaotic behavior justify the manufacturer’s claim?

### 3.1.5 Continuous system simulation languages and systems

For continuous systems modeling and simulation many diverse languages exist. Most of these are based on the modeling approach discussed above. They are said to adopt a *state-space description*. (There are also newer causal, non-state-space modeling approaches to continuous systems, e.g., Bondgraphs [Blundell 82, Thoma 90], see Chapter 9). Among the classical continuous simulation languages and systems we distinguish two major classes, namely,
languages based on the *Continuous System Simulation Language* (CSSL) standard [Augustin et al. 67] and

block oriented simulation systems.

We discuss these two approaches briefly in the following, considering two example implementations of the Van der Pol equations.

**CSSL simulation languages**
In CSSL, a continuous model is specified in language form by defining the set of differential equations of first order. Accordingly, there must be a statement to specify the integrator block. This is either done by an integrator expression or a derivative expression, depending on the language. Probably the most widespread CSSL language, ACSL [Mitchel and Gauthier 86], uses an integrator expression $\text{integ}(d, q0)$ with derivative and initial state as parameters. For example, the statement

$x = \text{integ}(v, x_0)$

says that the state variable $x$ is the integral over a variable $v$ and integration starts from initial value $x_0$.

Besides this essential statement, ACSL provides the usual expressions of a programming language in Fortran syntax and some special statements for simulation control. The program below shows the implementation of the Van der Pol model in ACSL. Such programs are divided into different sections. In the example, the first section is initialization section where some constants are defined. The main part is the DYNAMIC section and within it the DERIVATIVE section where the derivative equations are defined by integ-expression discussed above. Note that CSSL model texts are not programs in the standard sense. The model statements are not executed sequentially but instead they represent equations that are sorted and translated into an imperative, sequentially executed program.

```fortran
PROGRAM Van der Pol
  INITIAL
    constant
      k = -1, x0 = 1, v0 = 0,
      tf = 20
  END
  DYNAMIC
    DERIVATIVE
      x = integ(v, x0)
      v = integ((1 - x**2)*v - k*x, v0)
    END
    termt (t.ge.tf)
  END
END
```

**CSSL simulation model of Van der Pol equation**

**Block Oriented Simulation Systems**
Block oriented simulation tools work similarly to the programming of analog computers and are primarily used by control engineers. Modeling is done by coupling together primitive components and elementary functional building blocks. In a graphical user interface, blocks can be “dragged and dropped” into a model to form components in a network. The modeler then provides the interconnection information by drawing lines.
from output ports to input ports. Thus, here system specification is at the coupled model level as opposed to the state level represented by ACSL. 0 shows some elementary building blocks from Simulink [Mathworks 1996], a superset of the popular Matlab calculation package. Besides the blocks shown, the Simulink system provides many other blocks which also realize modeling concepts going beyond pure continuous modeling, in particular, building blocks for difference equations and blocks to realize discrete control systems. Our purpose here, however, is to discuss continuous network modeling by means of the Simulink system.

Once more, the most basic block is the integrator. It receives one real valued input and outputs the integral of the input trajectory starting from a particular initial state (a real value which is the first output — initial states are often called initial conditions in the differential equation literature). The integrator therefore realizes a memory element with a single state variable. Besides the integrator, a set of algebraic functions are provide. In 0 we see building blocks for the sum of two inputs, the multiplication of two inputs, and the gain element which amplifies the input by a constant c. There is also a function element which can be programmed in a C-like language. Sources are provided to generate test signals (e.g. Constant and sine generator).

Simulink allows hierarchical modeling by providing the subsystem block as a placeholder for a coupled component and the inport and outport blocks to model input and output of coupled systems.

![Figure 24 Some building blocks from the Simulink block oriented simulation package](image)

0 shows a block model specification of the Van der Pol equation. The inputs to the integrators are the outputs of a network of algebraic functions. These define the equations for the derivatives of state variables. The network generates the values of state variable x as its output.
3.2 Sources


