# 8 Background Information on Simulation

**Jan F. Broenink**
University of Twente, EE Department, Enschede, Netherlands

## 8.1 Introduction

Before presenting a general structure of the modeling and simulation process, first some terms and definitions are introduced. At the end of this section, alternatives to simulation are presented, together with reasons why simulation is a meaningful activity. Some shortcomings of simulation are also discussed.

### 8.1.1 Terms and definitions

Since a model is a simplified representation of the system under study, abstraction is inevitable to capture only the relevant and interesting aspects of the system. The basic modeling goal is to derive a *competent model*: a simple as possible model that is just as sufficient to give the information as requested in the goal of the model.

Competent simulation

In the same way, simulations should be as simple as possible, but extensive enough to give the answers requested. It is therefore useful to consider on beforehand:

- The *goal* of the simulations and the *assumptions* from which it can be deduced what simulations need to be performed.
- Expected *results* and the *ranges* wherein the variables of the model are valid. This can be used to judge the simulation results on its correctness.

Goal of simulation

The *goal of simulations* is in general (one of) the following:

- To *verify* the model, i.e. conclude from the simulation results whether the model looks correct.
- To *validate* the model, i.e. compare simulation results with real measurements.
- To compute numerical values of some model variables at given points in time.

Goal of model

Note that the goal of the model imposes demands on the simulations, both the specification and when the results can be qualified to be ‘good’. Therefore, we present some *general* goals of models:

- *Understand* / explain the dynamic behavior of the system under study. A rather detailed model with sophisticated resemblance to the physical behavior is needed.
- *Compute* a reaction in a control system of the system under study. A not so detailed model is needed: only the overall dynamic behavior is enough. Often a transfer function will do. Fast simulation is necessary when the model is used in a real controller.
- *Predict* the dynamic behavior of a system to be designed. Depending on the amount of design knowledge available, the level of detail of the model will vary accordingly.

Simulation Run

Multiple Run Simulation

Experiment

Simulation of a model on a digital computer means that the set differential equations describing the model are calculated by means of *numerical integration.* One *Simulation Run* is one simulation: time runs form $t_0$, the *start time* until $t_e$, the *end time*. We use this term, to distinguish it from the situation where time runs more times form $t_0$ to $t_e$. This latter case is called *Multiple Run Simulation*. The conducting of one or more simulation runs with a particular model is called *experimenting* with that model. All necessary data needed for simulations are called *experiment.*

Verification

By simulation, we can *verify* the model, i.e. check whether the model as written down in the simulation program behaves as we have designed to do so. For this check, we can run *test simulations* to judge the behavior of the model under particular circumstances, like equilibrium situation, step response, transient response. Important aspects to look at are the *shape* of the curves and the *value range* of the variables.

Using verification, we check with simulations if the model is competent or not.

Validation

Simulation results can also be compared with measurements on the real physical system. This is called *validation*. Using validation, we check with real measurements if the model is competent.

### 8.1.2 General structure of modeling and simulation process

It is clear, that after setting up the model, first verifications will be done, and after that validations if possible. When the model is rather complex, first parts are verified and after that the whole model is simulated in the verification process. If possible, also the validations can be done on the parts of the model (and system of course).

In a design situation, when the real system is not available yet, or when it is *not* possible to perform the measurements, verification *only* can indicate whether the model is competent.

When we are sufficiently satisfied on the model and its experiments, we can use the model for which we had made it. Like analysis of the system or deriving a controller. At least, a part of the modeling and simulation goals need to be satisfied.

The general structure of the modeling and simulation process is presented in Figure 8-1. Modeling and simulation is in principle iterative, because use of models in simulation and analysis will cause feedback on the correctness and meaningfulness of the model. In the initial phase, complex submodels can be modeled as simple elements, only representing the principal aspect. Refinements can be added in a later iteration (e.g. a gearbox is first modeled as an ideal transformer, while later moments of inertia and backlash are included).

As input for modeling and simulation the following items are needed:

- The model: *what* should be simulated
  - Description of the structure (equations, block diagram, bond graph, iconic diagram)
  - Parameter sets (values) and Initial Values of state variables
- The specification of a simulation run: *how* the simulation should be performed
  - Run specification: timing, integration method, multiple runs etc
  - Plot specification: what will be produced as output to the user: plots on screen, on file, animation etc

Separation of model and experiment

Model and experiment are separated here. The advantage is that when more experiments are done on the same model, we are sure that the *same* model is used. Furthermore, we separate the modeling process from the use of models (simulation, analysis), such that during modeling, we are not bothered by simulation strategies.

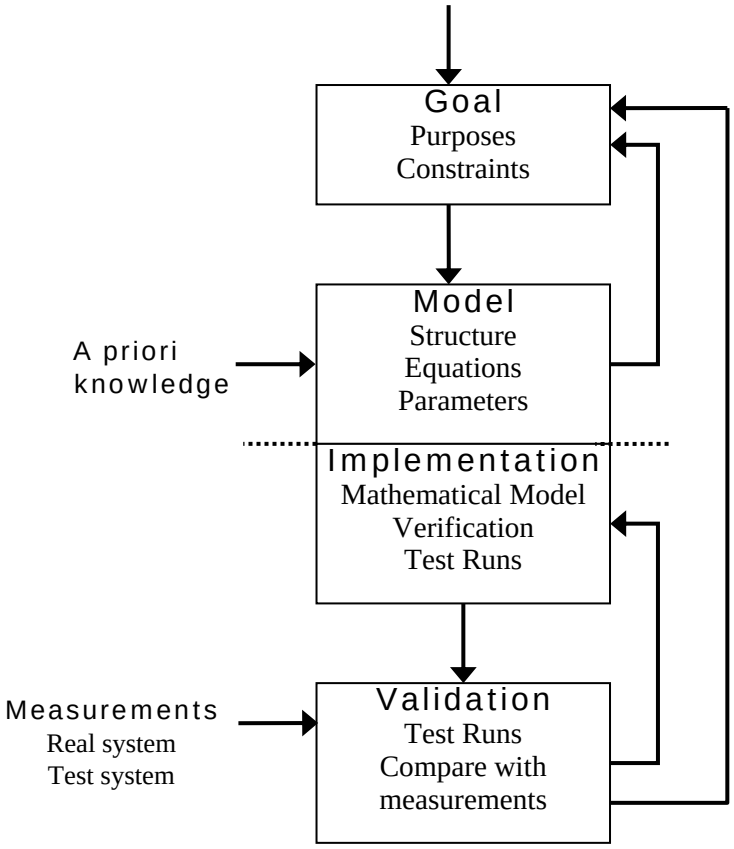


Figure 8-1 General structure of the simulation process

### 8.1.3 Other model types

Other model types

There are other types of models, which can be used to mimic the system under study. Examples are:

- *Scale models*, often used in a wind tunnel or towing tank, as used by the NLR (National Aerospace Laboratory in Amsterdam and N.O. Polder) and Marin (Maritime Research Institute Netherlands, in Wageningen).
- *Drawings* such as road maps and layouts of buildings. Software exists by which buildings are shown spatially, and a visitor can walk around virtually.
- *Facts and Rules* (if-then-rules), like diagnostic systems or driving instructions.

Scale models

Experiments with scale models are done to measure and model effects, which are not (yet) modeled on the right level of detail. Furthermore, (full) scale models are used to perform tests for quality control or safety of products. An example are crash tests with cars, ‘manned’ by dummy people stuffed with sensors, as is done at TNO Automotive’s Crash Center in Delft.

### 8.1.4 Usefulness of simulation

The usefulness and necessity of simulation are obvious: alternatives, like prototype tests or destructive tests, are really more expensive than simulation. Often, simulation is the only manner to obtain detained insight in the behavior, namely during design (there is not yet a prototype), critical situations (prototypes are too dangerous).

Advantages

The *advantages* of simulation are that results are faster available and more precise. Simulation is cheaper, and can be used as prediction. Furthermore, the assumptions must be made explicit, as well as the restrictions of the model are explicit.

Disadvantages

The *disadvantages* of simulation are that the results are of limited value: only the model itself is simulated, only the 'forgotten' aspects! Sometimes, results look promising and correct, but are not.

## 8.2 Background of numerical integration

### 8.2.1 History and basics

The use of digital computers for simulation only started around 1950. Before that, analogue computers were used: The mathematical operations were realized in hardware and grouped together to form a model by means of connections on a so-called *patch panel*. Later, the patch panels were substituted by a digital computer, resulting in a *hybrid computer*. Another means was the use of *equivalent physical models*, mostly electrical networks. This is an explicit use of analogies between phenomena in different physical domains.

For use on digital computers, rather soon simulation programs and simulation languages were developed, to get standard facilities available like numerical integration, and displaying models and simulation results. Refer to section 8.5 for an overview of some relevant simulation packages.

Time discretization

To compute the variables in the model as a time function on a digital computer, time will be discretized (see Figure 8-2). Otherwise, we have an infinite amount of values of time between $t_0$ and $t_e$, since $t$ is a real variable. Only at discrete points $t_k$ the model is computed. The result is that the model variables are available as a discrete series of values, and not as a continuous function. Such series are shown on a screen as dots. To give an idea of a continuous curve (of which the series of points is an approximation), straight lines interconnect the dots. This is in fact *linear interpolation*. In general this is accurate enough, because the distance between two adjacent points is small enough.

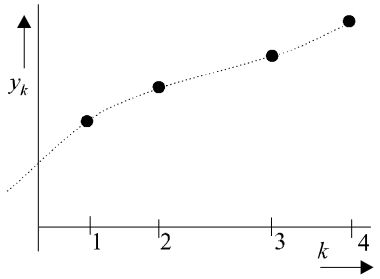


Figure 8-2: Discetization of time

*Time Step, $h_k$, Integration Interval*

The time distance between two adjacent points is called the *time step*, $h_k$, or *integration interval*. This $h_k$ need not necessarily be constant. As notation we use $x_k = x(t_k)$ in stead of $x(t)$. In the following we use the shorthand notation to denote a time-discretized variable.

Numerical Integration is an *approximation*

It is crucial to consider that numerical integration is an *approximation* of the real integration process. Besides errors due to the time discretization, errors are generated because numerical integration formulas are approximations of the real integration function. The solution generated by numerical solver has thus a certain *accuracy*. This implies that different numerical integration methods exist; each of them having their own class of models for which they are suitable. This means that we have to analyze both models as well as numerical integration methods on aspects relevant for simulation.

### 8.2.2 Simulation models

Initial Value Problem

The model suitable for simulation consists of a set of *Ordinary Differential Equations*, and is called a *simulation model*. These differential equations describe the system as a set of *state equations*, where the initial values of the state variables (also called *initial conditions*) are known. In numerical mathematics, this is called an *Initial Value Problem* (IVP)*:*

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \quad (1)$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, t) \quad (2)$$

State variables

Output variables

where **x** is the array of *state variables*, also called the state vector; **x**(0), the initial values, which are known. **u** is the array of input variables. **f** are the state equations. **y** is the array of *output variables*, computed from the states and inputs using the functions **g**.

Model equations

**y** and **g** are *not* necessary to compute the state of the model as function of time *t*. Only equation (1) is needed. The equations **f** and **g** together form the *model equations*.

Explicit model

Equations (1) and (2) denote an explicit model: all the information to compute $\dot{\mathbf{x}}$ is available. When, however, this is not the case, the model is implicit. This means that **f** contains one or more implicit equations. The general form of equation (1) changes to:

$$\mathbf{f}^{\#}(\mathbf{x}, \mathbf{u}, t) = 0 \quad (3)$$

ODE
DAE

If we rewrite equation (3) to a kind of explicit form, we get a set of ODEs with algebraic constraint equations, which is called a set of *Differential Algebraic Equations* (DAEs). Its most general form, including the update of equation (2) is:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}, t) \quad (4)$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}, t) \quad (5)$$

Implicit model

The occurrence of $\dot{\mathbf{x}}$ in the argument of **f** makes the model implicit. The set of output equations is basically the same, but also dependent of $\dot{\mathbf{x}}$.

Models containing implicit equations need special provisions in the numerical integration method, namely iteration. This topic is treated in the chapter on numerical mathematics (chapter 6). It is also possible to eliminate $\dot{\mathbf{x}}$ by rewriting the state equations. The implicit model *changes* to an explicit model. Explicit models mostly are easier and faster to simulate. However, interesting variables can disappear due to the rewriting process, and the accuracy of the simulation result can get really worse in some cases.

Stiff models
Undamped models

Another property of models relevant for simulation is its parameter values. *Eigen values*, $\lambda$, (or natural frequencies $\omega$ and relative damping $\zeta$) are a compact representation of the parameter values of the model. They can be used to determine if the model is *numerically stiff* (real parts of $\lambda$'s are far apart), or has little or no damping, (some $\lambda$'s have a very small real part). Stiff models need special numerical integration methods. Not every regular numerical integration method is suitable for models with undamped parts.

Discontinuities

The final issue of models to be considered is whether the model contains discontinuities. In other words, how smooth are the model equations, **f**? Theory on numerical methods regularly uses the assumption that **f** is continuous.

In the chapter on numerical mathematics (chapter 9), these model properties are treated.

### 8.2.3 Numerical integration methods

In the numerical integration method, the result of the model computation (equation (1) ) is used to compute the value of the states on the next time step: $x_{k+1} = x_{k+h}$. The discretization of the independent variable, *t*, is now

used. In Figure 8-3, a general scheme of the simulation process is presented. The simulation starts with the assignment of the initial values to the state vector $\mathbf{x}_0$. Then successive values of $\mathbf{x}_k$ and $\mathbf{y}_k$ (k=1, 2, 3, …) are calculated interatively by calculating the model equations and applying the numerical integration method alternatively (right-hand-side loop of Figure 8-3) until the end time $t_e$ is reached, or the simulation is interrupted by the user. Note that most numerical integration methods need calculation of the model equation (1) as a part of the integration process (left-hand-side loop in Figure 8-3).

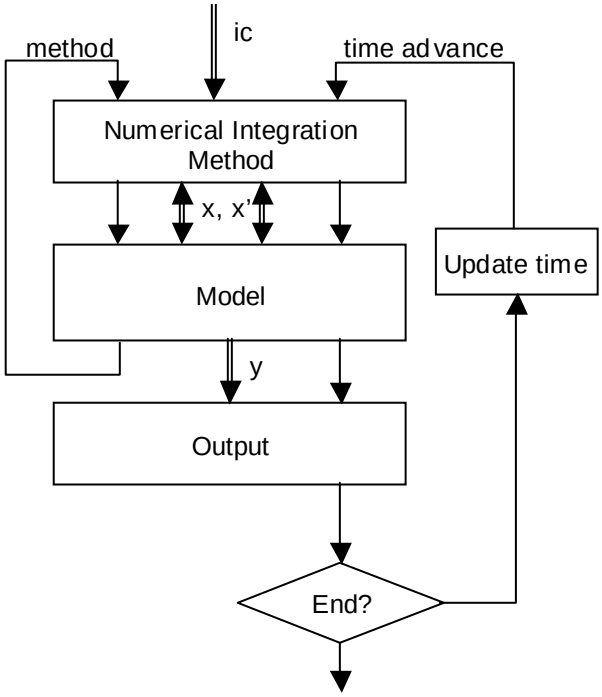


Figure 8-3: General Scheme of the simulation process

The general rule of a numerical integration method can be derived in different ways, starting from integration, or differentiation. Here we use differentiation:

$$\dot{x}(t) = \lim_{h_k \to 0} \frac{x(t+h_k) - x(t)}{h_k} \tag{6}$$

The limit operator cannot be used, because $t$ has been discretized. $x_{k+1}$ is approximated using previous values of $x$ and also using previous values of the derivative of $x$. The most simple numerical integration method is the Euler method: The time step $h = h_k$ is constant, and must be small enough to obtain enough accuracy. Euler has two forms:

Euler integration formula

$$x_{k+1} = x_k + h\dot{x}_k = x_k + hf(x_k, u_k) \tag{7}$$

$$x_{k+1} = x_k + h\dot{x}_{k+1} = x_k + hf(x_{k+1}, u_{k+1}) \tag{8}$$

Equation (7) is the explicit form; $x_{k+1}$ is computed from known data (everything on $t = t_k$ is known). Equation (8) is the implicit form: iteration is necessary to solve the equation.

General scheme

A numerical integration method specifies how $\mathbf{x}_{k+1}$ is approximated in terms of $\mathbf{x}$ itself and its derivatives, both at previous, current or future moments of time. Practical numerical methods consist of a linear combination of these terms $\mathbf{x}^{(i)}_{k+1-j}$ using parameters $\alpha_{ij}$, n and r ( $\dot{x} = x^{(1)}$ ):

$$\sum_{i=0}^{n} \sum_{j=0}^{r} \alpha_{ij} h^i_{k+1-j} \mathbf{x}^{(i)}_{k+1-j} = 0 \tag{9}$$

The method is explicit if all $\alpha_{i0}=0$ for i=1...n, otherwise the method is implicit. This equation is an approximation, so errors are made:

Truncation errors
Round-off errors

- *Truncation errors*, due to the non-idealness of the approximation formula.
- *Round-off errors*, due to the limited machine precision.

For a useful approximation, the time step $h_k$ depends on both the model and the integration method. Refer to the chapter on numerical mathematics (chapter 6) for further details.

Two main classes of integration methods exist:

- Multistep, first-order derivatives methods
  n=1, only **x** and its derivative are used. Higher order methods take more values of the past, until $x_{k+1-n}$.
- One-step higher-order derivatives methods
  r=1, only values of $\mathbf{x}_k$ and $\mathbf{x}^{(i)}_k$ are used. These methods are Runge-Kutta methods. No values of previous moments are used.

The more terms are in the numerical integration method, in general, the more accurate the method can be. Thus larger time steps, $h_k$, are possible with equal accuracy. Multistep methods are not really suited for models with discontinuities, because after each discontinuity, the integration process has to start as a first order method, and increasing the amount of previous values with one at each step.

Further elaboration on numerical integration methods and their properties takes place in the chapter on numerical mathematics (chapter 9).

## 8.3 Choosing a numerical integration method

For a particular model, a numerical method must be chosen such that the computational work during a simulation run is minimal and the calculated trajectories will have an error that is within a specified tolerance. In order to choose an optimal numerical integration method, global knowledge of some properties of the mathematical model is necessary:

- Presence of implicit equations
  Implicit models can only be simulated using a implicit method. Explicit can be simulated with both explicit and implicit methods.
- Presence of discontinuities
  After the discontinuity is located, simulation can proceed as if it started: it is again a initial value problem. Multi step methods are less accurate here. Fixed step methods can 'jump' over the discontinuities. Standard variable step methods are thus needed, although it is possible that a standard variable step method *jumps* over a discontinuity.
- Magnitude of the stiffness ratio S(t)
  S(t) is a measure of how far the real parts of the eigenvalues are apart from each other. Stiff models need stiff methods. The time step is determined by the accuracy instead of the eigenvalues. When the 'high' frequency components have decayed, the step size is no longer limited by them, and can grow until limited by the 'low' frequency components.
- Presence of oscillatory components
  These models have eigenvalues on the imaginary axis, i.e. have components *without* damping. These models should *not* be simulated with stiff methods, as the time step cannot be increased, because the 'high' frequency components do not decay. AM methods are a good candidate.

The following guidelines can be set up (Table 1). A brief explanation of the numerical integration methods is given in Table 2.

| Properties of mathematical model | Numerical integration method | |
|---|---|---|
| | Normal Case | Classroom Case |
| Explicit models | | |
| without discontinuities | RK-4 | AB-2, Euler |
| with discontinuities | RKF | RK-1(2) |
| stiff | BDF / Vode | Gear-2 |
| Implicit models | | |
| without discontinuities | BDF | Gear-2 |
| with discontinuities | GAM | Gear-2 |
| stiff | AM-2 | AM-2 |

Table 1 Guidelines for choosing a numerical integration method.

The full names of the methods are:

| | |
|---|---|
| Euler | Euler |
| RK-x | Runge-Kutta, $x^{th}$ order |
| RKF | Runge-Kutta-Fehlberg |
| RKDP8 | Runge-Kutta Dormand Prince, $8^{th}$ order |
| AB-x | Adams-Bashfort, $x^{th}$ order |
| AM-x | Adams-Moulton, $x^{th}$ order |
| GAM | Generalized Adams-Moulton |
| BDF | Backward Differentiation Formula |
| M-BDF | Modified Backward Differentiation Formula |
| Vode | Stiff ODE solver of Cohen & Hindmarsh |

| **Integration method** | **Type** | $\alpha_{i0}$ | $h_k$ | **Stiff?** | **Oscillatory?** |
|---|---|---|---|---|---|
| Euler | Single step | explicit | fixed | – | – |
| RK-x | RK type | explicit | fixed | – | + |
| RKF | RK type | explicit | variable | – | + |
| RKDP8 | RK type | explicit | variable | + | |
| AB-2 | Multistep: 2 | explicit | fixed | – | + |
| Gear | Multistep: 2 | implicit | fixed | + | + |
| AM-2 | Multistep: 2 | explicit | fixed | + | + |
| BDF | Multistep: 1-5 | implicit | variable | + | – |
| GAM | Multistep: 2 | implicit | variable | + | + |
| VODE | Multistep | explicit | Variable | ++ | + |

Table 2 Properties of numerical integration methods

## 8.4 Processing of models

Simulation model

A *simulation model* is a model suitable for simulation needs to be a set of sorted, computable *assignment statements*. A description in terms of a bond graph, block diagram or set of differential equations is a *functional* description. Each equation is valid for all values of the (problem) time *t*. The 20-SIM statement

$$FR = R * VM$$

actually means

$$\forall t : 0 \le t \le t_n : F_r(t) = R * V_m(t)$$

In Table 3, it is indicated which processing activities are needed to generate a suitable set of equations. These activities are:

- *Expansion*
  Removal of the hierarchical structure: on places where a reference to a (bond-graph) submodel is made the description itself is written.
- *Causal analysis*
  The computational form of the constitutive relations is determined

- *Rewriting equations*
  The equations are processed to assignment statements. This actually means *symbolically* rewriting of the equations to the desired form. Often the equations are optimized for simulation. This means that variables get eliminated.
  It is also possible to rewrite statement in such a way that an implicit model changes to an explicit model. Explicit models mostly are easier and fast to simulate. However, interesting variables can disappear due to the rewriting process, and the accuracy of the simulation result can get really worse in some cases.
- *Sorting*
  Ordering of the assignment statements in such a way that all terms at the right hand side are known when the statement will be calculated (states, parameters and constants are known before a model computation starts).

| Input | Hierarchical, acausal bond graphs | Hierarchical equations | CSSL specifications and macros | Block diagrams, causal bond graphs |
|---|---|---|---|---|
| Steps | Expansion<br>Causal analysis<br>Rewriting equations<br>Sorting | Expansion<br>-<br>Rewriting equations<br>Sorting | Expansion<br>-<br>-<br>Sorting | -<br>-<br>-<br>Sorting |
| Program | 20-SIM | Dymola/Modelica | ACSL | Simulink<br>TUTSIM |

Table 3 Processing steps

## 8.5 Simulation languages

Simulation languages can be *batch* oriented or can be *interactive*. Batch programs do not allow interaction during the processing and simulation. Examples are CSMP, CSSL, and ACSL. In early days this was the only available means for simulation. In interactive programs, often the model is *interpreted*, while the simulation results are presented *instantly* to the user. Interrupting the process on any moment is possible Examples are TUTSIM, PSI, 20-SIM, VisSim, SimuLink. Nowadays, most simulation packages are interactive. The current version of ACSL allows interactive simulation.

Of each group a representative is discussed, whereas 20-SIM is used for the lab sessions.

ACSL
A Continuous Simulation Language (Mitchell and Gauthier Associates). It uses the CSSL standard of the Society for Computer Simulation defined in 1967. Features are:

- The model consists of equations in a text file: special ACSL functions and FORTRAN.
- The model is translated to FORTRAN and linked with the ACSL run time library resulting in an executable program.
- During simulation some interactive facilities are present: changing parameter values, plot specifications
- Model, parameter values, and simulation control commands all are stored in one text file. The interactive simulation commands can also be read from a command file.

TUTSIM
Technical University of Twente SIMulation program, built at the chair of Control Engineering (Electrical Engineering) of UT. The program exists

since 1970, and is now marketed by Meerman Automation in Europe and by TUTSIM Products in the USA.

- Models in form of causal bond graphs or block diagrams can be specified by means of a "structure table" straightforwardly.
- The model is interpreted. Submodels are not possible
- Modeling and simulation are instantly interactive, which was rather an unique feature at that time.
- Model, parameter values, and specification of the run are stored in *one* binary text file

Dymola / Modelica

DYnamic Modeling Laboratory, derived at the Control Department of the Technical University of Lund, Sweden. It is marketed and further developed by Dynasim in Lund.

- Models in form of acausal equations. The graphical editor is a look alike of Simulink.
- The model is compiled to DSBlock, and linked with a simulation library to an executable simulation program.
- Plots are shown after the simulation is finished, using a separate viewer program.
- Model, parameters and run specification are all stored in *one* text file. Plot specifications are stored separately as command files.

Its successor is called Modelica, where an object–oreinted approach for modeling gets emphasized. The development of Modelica is done in a European wide design team, in order to work towards a standard modeling language for physical system models. This activity now gets widely recognized. See also http://www.Modelica.org

Simulink / Matlab

Matlab developed in the early 1980s, is being marketed and developed by the MathWorks Inc, Natick, MA, USA. It started as a command-line based program for easy calculation and graphic display of the results. Main focus was numerical mathematics and signal processing. Later, Simulink came, a graphical editor for block diagrams. Nonlinear systems are now really supported.

Currently, the company has more than 1000 employees, and is the world market leader for technical computing and model-based system design.

20-SIM

TwenteSim, developed at the chair of Control Engineering (Electrical Engineering) of the University of Twente. Its former name was CAMAS. The program exists since 1990. It is now being marketed and further developed by Controllab Products B.V., a spin-off company of the UT.

- Models are entered graphically as acausal bond graphs, iconic diagrams, block diagrams or as sets of equations.
- Hierarchical models are translated to a simulation model, where formula manipulation is used to rewrite the equations to the correct causal form.
- Simulations are interactive, with possibilities for animation using Direct-X libraries. It is a really fast simulator.
- Model, parameter values and specification of the run are stored in separate text files (version 2.X). In version 3.X the model and experiment are stored in different binary files.

Currently version 3.3 is on the market. In version 3.X, the ideas of object-oriented physical systems modeling are implemented. Furthermore, multibonds (more dimensional bonds and submodels), animation, iconic diagrams, and units are added. 20-SIM has also support for controller design, competing MATLAB, but 20-SIM has a link to MATLAB at different levels (model, signals, data), See also http://www.20sim.com