# SOLVING DIFFERENTIAL EQUATIONS IN TRNSYS WITHOUT PROGRAMMING

Werner Paul Keilholz[1], Peter Riederer[1], and Vanessa Ducreux[1]
[1]CSTB Sophia Antipolis, France

## ABSTRACT

In building simulation tools differential equations are widely used to model physical phenomena of components such as walls, air and any kind of system component in the building. Especially when simulation is used to study and optimise system control, the models used are mainly transient models solving differential equations in order to represent correctly the transient behaviour of the whole control loop.
The TRNSYS simulation environment is a powerful tool allowing the simulation of a large number of problems. However, developing and/or implementing new models (called 'Types' in TRNSYS terminology) can be an obstacle for those users who use TRNSYS as a simulation environment and not a modelling and simulation environment. In most simulation studies, there is a need for simple models considering phenomena that are specifically important for the actual problem. Simple steady state models on the one hand can easily be implemented by the TRNSYS equation type or by calling external programs such as Excel etc. Dynamic models on the other hand can only be implemented by writing new types in a programming language such as C, C++, Fortran, Visual Basic, etc., or by coupling to other environments. In order to overcome this barrier, a new type has been developed allowing implementing and calculating, without any programming, a dynamic model.
With this model, it shall be possible to define simple dynamic models such as sensors, drives, furniture, specific walls, etc.

## APPROACH

To allow for differential equation systems, the development has been applied to a matrix based approach. This will allow for the modelling of more complex systems.

Many heat transfer and storage phenomena can be modelled by the state space approach:

$$\dot{X} = AX + BU \tag{1}$$

$$Y = CX + DU \tag{2}$$

where A and C are i,i matrices, and B and D i,j matrices, X the state vector, U the input vector and Y the output or observation vector. In the developed type, only equation (1) is implemented to date.

Our goal is to compute x for given initial values of *X*, the input vector U and the matrices A and B. Examples for phenomena easily modelled by such an equation include liquids flowing in ducts, adjacent volumes of air, volumes of ground in a 2 or 3 dimensional ground model, etc. In terms of TRNSYS components, we would like to be able to compute the state vector *X* from the input vector *U*, the initial conditions for the X-values, as well as the matrices A and B (stored in external files).
In order to be able to model also non-linear systems, the state space equation has been modified as follows:

$$\dot{X} = A_{cst} \cdot X + A_{\mathrm{var}1} \cdot In_1 \cdot X + \ldots A_{\mathrm{var}n} \cdot In_n \cdot X + B_{cst} \cdot U + B_{\mathrm{var}1} \cdot In_1 \cdot U + \ldots B_{\mathrm{var}n} \cdot In_n \cdot U \tag{3}$$

with

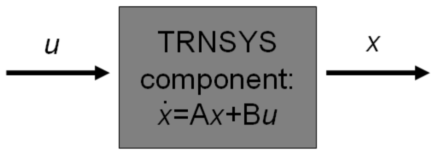
$$A = A_{cst} + A_{\mathrm{var}1} \cdot In_1 + \ldots A_{\mathrm{var}n} \cdot In_n \tag{4}$$

$$B = B_{cst} + B_{\mathrm{var}1} \cdot In_1 + \ldots + B_{\mathrm{var}n} \cdot In_n \tag{5}$$

As shown in equations (3)-(5), the matrices A and B are divided into two parts, one with constant values and one with variable values. This will allow for example the simulation of a pipe where a changing flow rate will need the recalculation of the matrices A and B.
Before integrating the derivative of the state vector, the "final" matrices A and B will be calculated at each time step following equations (4) and (5).

## IMPLEMENTATION IN TRNSYS

Once the matrices have been calculated, they can be loaded into the TRNSYS type. In the TRNSYS block diagram, the type will be shown as in the figure below:



The TRNSYS simulation environment provides two main mechanisms allowing the user / programmer to solve differential equations:

a) an approximate analytical solution (using the DIFFERENTIAL_EQN() function, implemented in the TRNSYS kernel)
b) a numerical solution (using the T and DTDT arrays passed to all components as an argument)

While b) is more powerful (as it is able to solve a wider range of problems, using any form of differential equation), a) is usually quicker. We chose to use approach a) in our solution.
The DIFFERENTIAL_EQN() function provided by the TRNSYS kernel requires the equation to be of the form :

$$\dot{X} = AA\,X + BB \qquad (6)$$

with

$$AA = A \qquad (7)$$

$$BB = B \cdot U \qquad (8)$$

While the matrix AA is identical to A, the matrix BB is the product of the matrix B as defined in the previous section multiplied with the input vector.

The DIFFERENTIAL_EQN() function is only able to treat one value of X at a time. In order to apply it to (6), two steps are required: first, the values of the *U* vector must be introduced. This can be achieved by simply multiplying B and *U* before solving the differential equation – *U* is thus 'included' in *BB*. Second, (6) must be computed for each of the i lines of *X*. As the lines are interdependent, this process must be repeated in an iterative way, until all results for X are identical to the results obtained during the previous iteration.

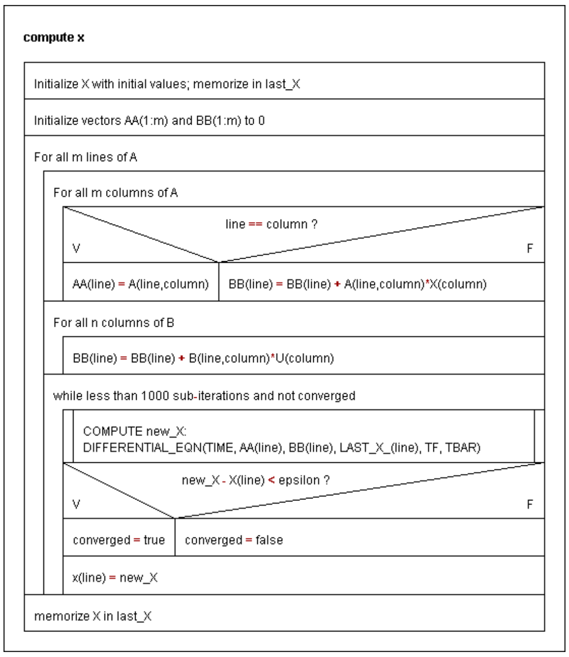
## <u>THE ALGORITHM</u>

The Nassi-Schneiderman diagram shows the algorithm used in the new type. In addition to the matrices A and B, it uses two vectors, AA and BB, which allow computing the i differential equations one by one. AA is initialized with the diagonal of A, and BB with sum of A(i,j)*X(j) for each line i. All elements of BB are then multiplied with their respective u value, to account for the term *BU* in (1) which is not in (6).

The type then performs a sub iteration loop until either convergence is reached (none of the newly computed X values differs from the ones computed during the previous sub iteration by more than epsilon, which is currently set to 1E-6), or the maximum number of iterations (fixed to 1000 for now) is reached. The number of iterations used is an output of the type, so that the user can take action in case convergence is not reached in too many time steps.

At each sub iteration step, DIFFERENTIAL_EQN is called to compute a new X value for the current equation. During sub iterations, TBAR (the mean value of the wanted variable over the timestep) is used, while after conversion we use the TF value returned by DIFFERENTIAL_EQN, which is the value at the <u>end</u> of the time step. This method proofed more precise.

The resolution strategy is shown in the figure below.



```
compute x
Initialize X with initial values; memorize in last_X
Initialize vectors AA(1:m) and BB(1:m) to 0
For all m lines of A
    For all m columns of A
        line == column ?
        V: AA(line) = A(line,column)
        F: BB(line) = BB(line) + A(line,column)*X(column)
    For all n columns of B
        BB(line) = BB(line) + B(line,column)*U(column)
    while less than 1000 sub-iterations and not converged
        COMPUTE new_X:
        DIFFERENTIAL_EQN(TIME, AA(line), BB(line), LAST_X_(line), TF, TBAR)
        new_X - X(line) < epsilon ?
        V: converged = true
        F: converged = false
        x(line) = new_X
memorize X in last_X
```

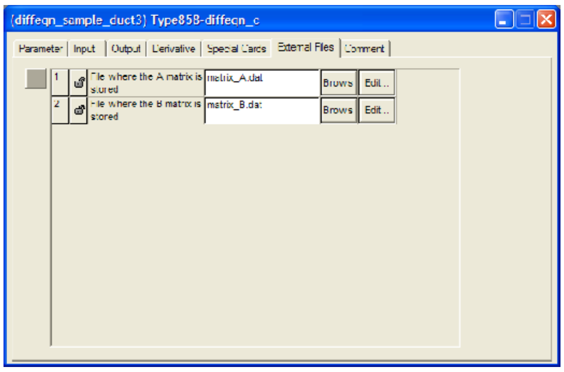
## <u>PRACTICAL IMPLEMENTATION</u>

The matrices A and B (with their constant and variable parts respectively) have to be saved in two files, one for A and for B. The calculation of the cells can be done by hand or in other programs, for example Matlab, as in the application example in the next section. The advantage of this approach is that it is possible to simulate exactly the same system in Matlab and in TRNSYS, without a new calculation or reconfiguration. All transient models in the SIMBAD library (an HVAC toolbox for the MATLAB/Simulink used at CSTB) for example are based on the state space approach, some of them can thus be easily used in TRNSYS, just by saving the matrices A and B in files.

The file has to follow the following structure in the case of the matrix A:

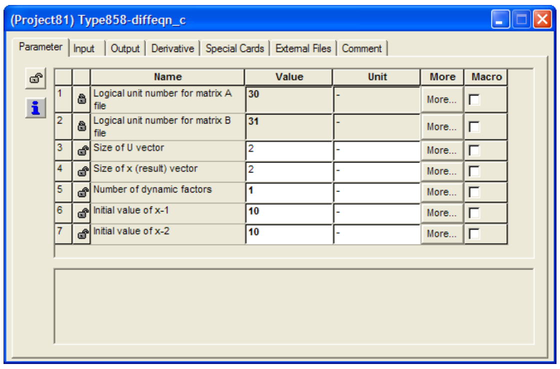
| | | |
|---|---|---|
| Acst(1,1) … | Acst(1,i) … | Acst(1,m) |
| Acst(2,1) … | Acst(2,i) … | Acst(2,m) |
| | | |
| Acst(m,1) … | Acst(m,i) … | Acst(m,m) |
| | | |
| VAR 1 | | |
| Avar(1,1) … | Avar(1,i) … | Avar(1,m) |
| Avar(2,1) … | Avar(2,i) … | Avar(2,m) |
| | | |
| Avar(m,1) … | Avar(m,i) … | Avar(m,m) |

The file for matrix B has to be defined in the same way.
In TRNSYS, both files are loaded in the “External files” menu of the type.



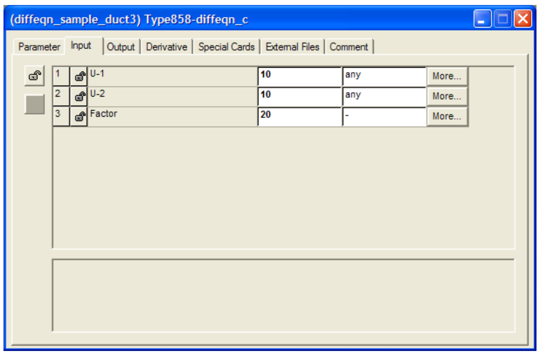
| | | | | | |
|---|---|---|---|---|---|
| 1 | | File where the A matrix is stored | matrix_A.dat | Brows | Edit... |
| 2 | | File where the B matrix is stored | matrix_B.dat | Brows | Edit... |

In the “Parameter” menu of the type, the user has to set the size of the state vector X, the size of the input vector U, the number of inputs that modify the matrices A and B as well as the initial values of the state vector.



| | | Name | Value | Unit | More | Macro |
|---|---|---|---|---|---|---|
| 1 | | Logical unit number for matrix A file | 30 | - | More... | ☐ |
| 2 | | Logical unit number for matrix B file | 31 | - | More... | ☐ |
| 3 | | Size of U vector | 2 | - | More... | ☐ |
| 4 | | Size of x (result) vector | 2 | - | More... | ☐ |
| 5 | | Number of dynamic factors | 1 | - | More... | ☐ |
| 6 | | Initial value of x-1 | 10 | - | More... | ☐ |
| 7 | | Initial value of x-2 | 10 | - | More... | ☐ |

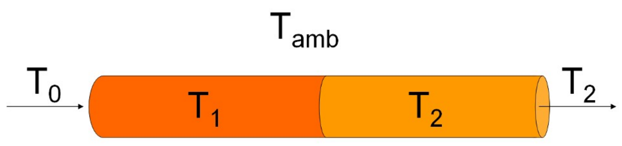
Finally, the input menu of the type allows the connection of the input values to the state space equation as well as the factors that have to be multiplied with the matrices.



| | | | | | |
|---|---|---|---|---|---|
| 1 | | U-1 | 10 | any | More... |
| 2 | | U-2 | 10 | any | More... |
| 3 | | Factor | 20 | - | More... |

## AN APPLICATION EXAMPLE

A typical phenomenon obeying equation (1) is liquid flow through a pipe. A liquid, characterized by its specific heat *cp*, enters the pipe at a temperature $T_0$. In order to model the temperature distribution inside the pipe, we define m equal pipe segments to which we assign temperatures $T_1$, $T_2$, … $T_m$.
To illustrate the method we choose m = 2 (2 pipe segments) for our simple example.
The pipe is placed in an environment at an ambient temperature $T_{amb}$, which causes heat losses between the pipe and the ambient, with a heat transfer coefficient h.
We assume the surfaces A of each segment to be equal, as well as the mass m of fluid inside a segment. The flow rate passing through each segment is called $\dot{m}$ .



The equation for the conservation of energy for the first segment (the one with temperature $T_1$) in this configuration yields:

$$m * c_p * \frac{dT_1}{dt}$$
$$= \dot{m} * cp * (T_0 - T_1) + h * A * (T_{amb} - T_1) \qquad (9)$$

Similarly, the energy balance for the second segment yields:

$$m * c_p * \frac{dT_2}{dt} =$$
$$\dot{m} * cp * (T_1 - T_2) + h * A * (T_{amb} - T_2) \quad (10)$$

With respect to equation (1), the temperatures T$_1$ and T$_2$ represent the state vector X (the variables we seek to compute). To obtain the matrices A and B, we have to transform equations (9) and (10) so as to obtain the form of equation (1):

$$\frac{dT_1}{dt} = (-\frac{\dot{m}}{m} - \frac{h*A}{m*cp}) * T_1 + \frac{\dot{m}}{m} T_0 + \frac{h*A}{m*cp} * T_{amb} \quad (11)$$

$$\frac{dT_2}{dt} = \frac{\dot{m}}{m} * T_1 + (-\frac{\dot{m}}{m} - \frac{h*A}{m*cp}) * T_2 + \frac{h*A}{m*cp} T_{amb} \quad (12)$$

**Modelling with constant flow rate**

With $\frac{dT}{dt} = \dot{X}$ and T = X this can be written

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{bmatrix} -\frac{\dot{m}}{m} - \frac{h*A}{m*cp} & 0 \\ \frac{\dot{m}}{m} & -\frac{\dot{m}}{m} - \frac{h*A}{m*cp} \end{bmatrix} * \begin{pmatrix} T_1 \\ T_2 \end{pmatrix} + \begin{bmatrix} \frac{\dot{m}}{m} & \frac{h*A}{m*cp} \\ 0 & \frac{h*A}{m*cp} \end{bmatrix} * \begin{pmatrix} T_0 \\ T_{amb} \end{pmatrix} \quad (13)$$

Still with respect to (1), we thus find

$$A = \begin{bmatrix} -\frac{\dot{m}}{m} - \frac{h*A}{m*cp} & 0 \\ \frac{\dot{m}}{m} & -\frac{\dot{m}}{m} - \frac{h*A}{m*cp} \end{bmatrix} \text{ and}$$

$$B = \begin{bmatrix} \frac{\dot{m}}{m} & \frac{h*A}{m*cp} \\ 0 & \frac{h*A}{m*cp} \end{bmatrix} \quad (14)$$

This example allows the simulation of the pipe assuming constant flow rate. However, in many practical cases the flow rate can change throughout operation and thus it should also be possible to vary it during the simulation.

**Modelling with variable flow rate**

Based on the same equations it is also possible to consider for a varying flow rate, thus the nonlinear case of the system.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \left\{ \begin{bmatrix} -\frac{h*A}{m*cp} & 0 \\ 0 & -\frac{h*A}{m*cp} \end{bmatrix} + \begin{bmatrix} -\frac{1}{m} & 0 \\ \frac{1}{m} & -\frac{1}{m} \end{bmatrix} * \dot{m} \right\} * \begin{pmatrix} T_1 \\ T_2 \end{pmatrix}$$
$$+ \left\{ \begin{bmatrix} 0 & \frac{h*A}{m*cp} \\ 0 & \frac{h*A}{m*cp} \end{bmatrix} + \begin{bmatrix} \frac{\dot{m}}{m} & 0 \\ 0 & 0 \end{bmatrix} * \dot{m} \right\} * \begin{pmatrix} T_0 \\ T_{amb} \end{pmatrix} \quad (15)$$

with the matrices A:

$$A_{cst} = \begin{bmatrix} -\frac{h*A}{m*cp} & 0 \\ 0 & -\frac{h*A}{m*cp} \end{bmatrix} \text{ and}$$

$$A_{var} = \begin{bmatrix} -\frac{1}{m} & 0 \\ \frac{1}{m} & -\frac{1}{m} \end{bmatrix} \quad (16)$$

and the matrices B:

$$B_{cst} = \begin{bmatrix} 0 & \frac{h*A}{m*cp} \\ 0 & \frac{h*A}{m*cp} \end{bmatrix}$$

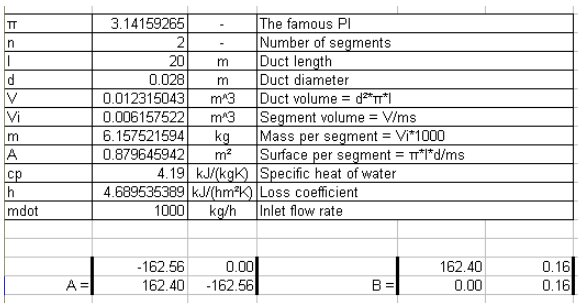
and

$$B_{var} = \begin{bmatrix} \frac{1}{m} & 0 \\ 0 & 0 \end{bmatrix} \quad (17)$$

The whole system can also be programmed in other tools such as Excel, Matlab, Scilab, Python or any other environment.

## NUMERICAL EXAMPLE AND VALIDATION

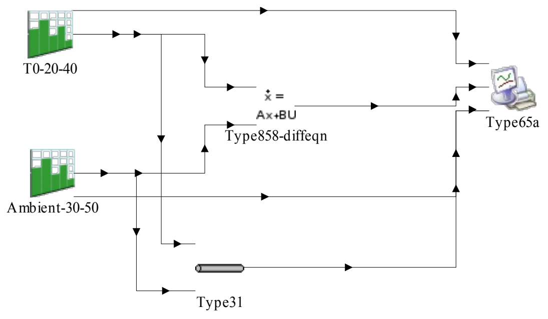
In order to compare our pipe model to the existing TRNSYS TYPE 31, we choose the default parameters of this component, but with a flow rate of 1000 kg/h, a length of 20 m and a diameter of 0.028 cm. A simple spreadsheet allows us to compute A and B:



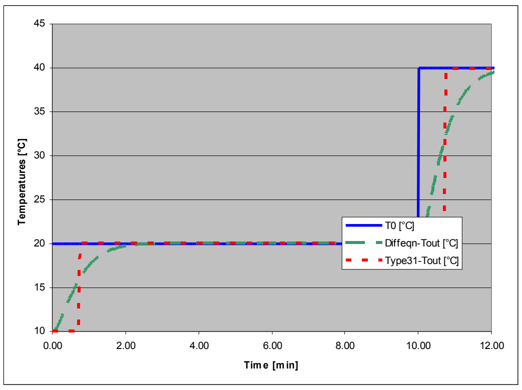
| | | | | | |
|---|---|---|---|---|---|
| π | 3.14159265 | - | The famous PI | | |
| n | 2 | - | Number of segments | | |
| l | 20 | m | Duct length | | |
| d | 0.028 | m | Duct diameter | | |
| V | 0.012315043 | m^3 | Duct volume = d²*π*l | | |
| Vi | 0.006157522 | m^3 | Segment volume = V/ms | | |
| m | 6.157521594 | kg | Mass per segment = Vi*1000 | | |
| A | 0.879645942 | m² | Surface per segment = π*l*d/ms | | |
| cp | 4.19 | kJ/(kgK) | Specific heat of water | | |
| h | 4.689535389 | kJ/(hm²K) | Loss coefficient | | |
| mdot | 1000 | kg/h | Inlet flow rate | | |
| | | | | | |
| | | | | | |
| | -162.56 | 0.00 | | 162.40 | 0.16 |
| A = | 162.40 | -162.56 | B = | 0.00 | 0.16 |

We simply copy-paste the values computed for A and B into the 2 external files of our type (which are simple text files), and the model can be used in a TRNSYS *Simulation Studio* project.

The figure shows a project designed to compare our home made type and standard duct/pipe type 31: two forcing functions provide assumptions for inlet temperature $T_0$ and ambient temperature $T_{amb}$ (which has practically no influence in the default configuration). Initial pipe temperature is 10°C , the water inlet temperature is 20°C. After 10 minutes, the inlet water temperature rises to 40°C (still using the same flow rate of 1000 l/h).

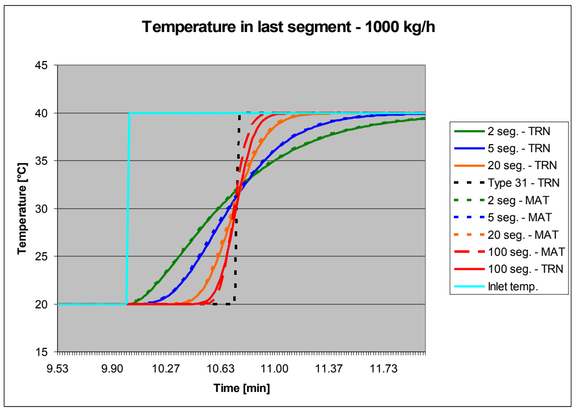


The same values are fed into both types, and the result is displayed. The resulting curves allow to clearly distinguish the different approaches used by the two models: while TYPE 31 uses a more sophisticated approach of dynamically segmenting the pipe, our model uses a constant number of segments (2 in the example, but this number can of course be easily increased).

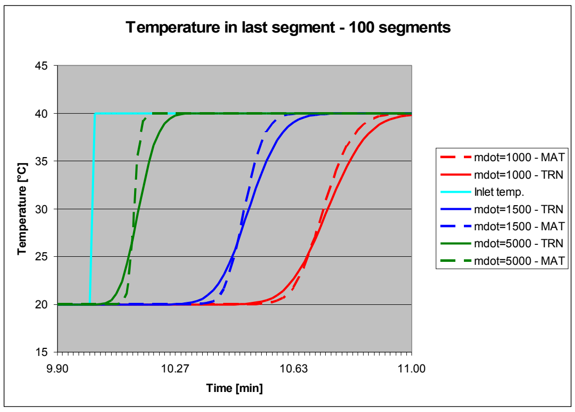


The result is as expected: the simple two zone model is less accurate than type 31. The delay in the outlet temperature following the step is well represented in type 31 while the simple two zone model is not able to calculate this delay. The result of the simple model can be improved by choosing a higher number of volumes in order to consider this delay and also to represent the dynamics at the pipe outlet more realistically.

Increasing the number of segments increases the accuracy, as expected. The figure below shows temperature curves for 2, 5, 20 and 100 segment models. The model has been implemented both in MATLAB and TRNSYS using the new type, based on the same equations, time step (1 second) and initial conditions in both cases:



This study attracted our attention to slight differences between the MATLAB and TRNSYS versions of the model, which become more obvious as the number of segments increases (while one would expect the model to become more and more precise).
Further investigation has shown that our model seems to become inaccurate for systems with low time constants: we can clearly see the phenomenon for higher flow rates (such as 5000 kg/h), decreased duct length and/or duct diameter, etc. and bigger time steps (we use 1 second). A parametric study varying the flow rate has been done:



This phenomenon may require further study (and care when using the current version of the model under the before mentioned conditions).

## PERSPECTIVES AND FUTURE WORK

Our simple example was only indented to illustrate the concept of our approach; we are well aware that our little model built in the demo application described above will not replace standard type 31. The approach can, however, be very interesting to model simple components such as temperature sensors, furniture, walls or also more detailed models such as geothermal boreholes with a three dimensional approach, etc. In such cases, our type could serve domain experts to experiment new mathematical models based on the differential equation $\dot{x} = Ax + Bu$ in TRNSYS, without programming.
The main advantage is that if a model is built in such a way in any other environment, it can be used without modification in TRNSYS. Models built in Matlab or Simulink can now be used in TRNSYS allowing to combine fast simulation speed of TRNSYS with easy modelling of Matlab. In addition to potential gains in simulation speed (which was not our main motivation in this work), the final user does not need to have access to the environment originally used to create the model (unlike the approach using the current TRNSYS type 155, which executes MATLAB at each iteration).

To help the user decide if this model could be an interesting option, we summarize the different existing approaches, together with their main characteristics, below:

- Type 155: Allows to easily reuse any existing MATLAB model. Requires programming in MATLAB to create a new component. Requires the final user to install MATLAB.
- Our new type: Allows to easily reuse existing MATLAB models based on a differential equation Ax+BU. Allows to easily create new components based on this equation without any programming. Does not require the final user to install any third-party software.

Using existing SIMULINK models has drawbacks even when using type 155, as the model would have to be called from MATLAB (at each timestep). Other, more advanced coupling methods are available for this type of model, as described in our article “Coupling of TRNSYS with Simulink – a method to automatically export and use TRNSYS models within Simulink and vice versa”.

Further validation on models created with our new component should be done, especially when many variables are tightly coupled. It would be particularly interesting to observe the evolution of computation time in these cases – in our simple example with two variables, we only need 2 sub-iterations at the worst, and in our trivial example the new type is about 2 times slower than type 31 when using 20 segments for both models. It would be interesting to test more complex scenarios.

The model is currently configured to allow for 200x200 matrices, in order to keep the size of the input and output vectors reasonable (and manageable in a graphical environment such as Simulation Studio). If the concept should proof useful, the observation equation can be added in order to output only the variables that are necessary and to limit the connections in the TRNSYS Simulation Studio.

## ACKNOWLEDGEMENT

This article has been elaborated in the frame of the DYNASIMUL project funded by ANR.

## REFERENCES

MATLAB: http://www.mathworks.com/
TRNSYS: http://software.cstb.fr

Bronstein, Semendjajew: Taschenbuch der Mathematik; Thun, Frankfurt/Main, 1987

Riederer P., Keilholz W., Ducreux V.. Coupling of TRNSYS with Simulink – a method to automatically export and use TRNSYS models within Simulink and vice versa. BS2009, Glasgow, 2009.