Robustness of Component Models in Energy System Simulators

Brian Elmegaard
Department of Mechanical Engineering
Technical University of Denmark
DK-2800 Kgs. Lyngby
Denmark
E-mail: be@mek.dtu.dk

Abstract

During the development of the component-based energy system simulator DNA (Dynamic Network Analysis), several obstacles to easy use of the program have been observed. Some of these have to do with the nature of the program being based on a modelling language, not a graphical user interface (GUI). Others have to do with the interaction between models of the nature of the substances in an energy system (e.g., fuels, air, flue gas), models of the components in a system (e.g., heat exchangers, turbines, pumps), and the solver for the system of equations. This paper proposes that the interaction between models and solvers should be made more robust by making more robust component models which handle the exceptions of the equations inside them, instead of or in addition to trying to remedy these by adding features to the numerical solvers. Improving the robustness of a component model may be more or less difficult depending on the mathematical expressions creating the exceptions. The proposed idea suggests that the solvers may be helped by exception handling leading it back on the right track. The original equation of the model is substituted by an algorithm, so the original equation is only evaluated where it is defined. Outside this region an algorithm is introduced, so the model iterates back to the feasible region. It is shown how this can be done for four different model of energy system component models: turbine constant, gasifier, heat exchanger effectiveness, and heat exchanger heat transfer coefficient.

Nomenclature

A	Heat transfer area
B	Number of atoms of an element per mole of a gas compound
c	cold side (index)
C_T	Turbine Constant
δ	A small number
ε	effectiveness
G	Gibbs free energy
g^0	Standard Gibbs free energy
h	Enthalpy
h	hot side (index)
i	inlet (index)
ΔT_{lm}	Log mean temperature difference
lim	limit (index)
$\dot{m}$	mass flow
max	maximum (index)
min	minimum (index)
$\dot{n}$	Molar flow
o	outlet (index)
p	pressure
$\dot{Q}$	Heat flow
R	Gas constant
T	Temperature
U	Heat teansfer coefficient
x	Variable in example
y	Molar fraction
z	Variable in example

1 Introduction

Many solvers for energy system simulation are available commercially, inhouse in companies or as more or less experimental tools in universities (Refer to [2, 5, 7, 9, 13, 10] for lists of such tools.). A tool of this type will (usually) include:

- input-output facilities
- model checking
- model solving
- models of thermodynamic properties of fuels and fluids
- models of energy system components and/or possibilities for implementing these

The tools may be distinguished by several features:

- equation-based versus component-based modelling
- text-based versus graphical user interface
- solvers for algebraic versus differential equations
- solvers based on (sequential) functional iteration versus (simultaneous) numerical solvers

It is probably a matter of the selected implementation of a tool (and a matter of taste of the user and the developer) which methods are claimed to be most efficient and user-friendly in model implementation and simulation[1].

1.1 Exceptions in Mathematical Formulation of Models

In the modeling process[11], each step from the mind model, through physical and mathematical models to numerical model, may introduce exceptions where the model is not well-defined for given values of some of the variables. For instance, a negative value of pressure does not make physical sense, but may unintentionally be applied to the numerical model during iterations, and thus should be handled.
Several methods for taking care of the problems simulation of a model may cause, have been proposed and are in use in different software. Most of the methods handle the problems by catching the exceptions and issue an error message. The ways that may be followed is to some extent prescribed by the way the system of equations is represented internally in the software. If the model is written in plain text much more information about the equations is available compared to applications where models are compiled code. For the latter case, Morton an Collingwood [14] describes a way to numerically analyze a system of equations to determine problems in its initialization.

It should be mentioned that several methods with better convergence characteristics than the ordinary Newton method have been proposed, e.g., Powell's method[3] and continuation methods [1], but these all seem to require continuity of the equations.

In the software EES (Engineering Equation Solver)[12], (which is equation-based rather than component-based), the user may define limits on all variables to keep them in the feasible region. In connection with a graph analysis of the system of equations to solve smaller systems in sequence, a robust solving environment is available. This method, however not well documented in literature, seems to be similar to the method of Shacham and Brauner [16], which, after thorough analysis of the equations, introduces subexpressions of all equations with discontinuities and controls the variables to stay inside these boundaries or steps inside another region. These methods both require special techniques to be used in the solver, whereas this paper proposes that the problems created by the model should also be handled by the model.

2 Proposed Idea

The above described methods catch some exceptions and remedy the problem or warn the user about it. One problem that is not handled in a way that ensures convergence of the simulation, however, is the problem connected to some variables reaching values that makes the model be evaluated where some equations are mathematically undefined. In the present paper, it is proposed that models may be extended such that the exception will not result in an evaluation of the model at such points, but instead of a modified model which makes the values of the variables return to the well-defined range through iterations.

[1] *Simulation* is in this paper referred to as the act of solving a system of equations. A dynamic system may involve differential equations, and will as such be a number of "simulations" solved in sequence.

3 Examples of Application

This section describes a number of applications of the proposed methodology, where it has shown useful during the implementation of DNA (See appendix for an description of the program). As a first simple example the natural logarithm, defined as $y = ln(x)$ has been implemented robustly in EES as a residual, as shown in Algorithm 1.

Algorithm 1 Making the natural logarithm robust

if $x > 0$ **then** {x is positive and the logarithm is defined}
  $0 = \ln(x) - z$
else if $z = 0$ **then** {Makes the solver converge also for $z = 0$}
  $0 = x - 1$
else if $z > 0$ **then** {Iterates to a positive x for $x < 0$ and a $z > 0$}
  $0 = x \cdot z - 1$
else {Iterates to a positive x for $x < 0$ and a $z < 0$}
  $0 = x \cdot z \cdot z - 1$
end if

The idea is that if the iterations is trying to calculate the residual for a negative value of x, where it is undefined. The exception is caught by defining branches to the residual, which will not have a solution in the interval where they are defined, but instead will lead x back to the positive axis. This implementation has been tested and found to converge for x defined and z dependent and vice versa, for x-values in the interval $[10^{-5}; 10^{5}]$ and different guesses on z and x making the calculation start in each of the branches. For low values of z, a factor may be multiplied to the first term in the last two branches and a term higher than 1 may be subbtracted. The branch for $z = 0$ seems to be superfluous, but in the present EES implementation it has been found necessary.

3.1 Turbine Constant

A turbine, as shown schematically in Figure 1, produces power by allowing a compressed gas to expand and by the kinetic energy from this expansion make the turbine blades move. It is commonly modelled by an isentropic or polytropic efficiency and by a turbine constant, which relates mass flow to pressure according to the formulation:

$$C_T = \frac{\dot{m}\sqrt{T}}{\sqrt{p_i^2 - p_o^2}} \tag{1}$$

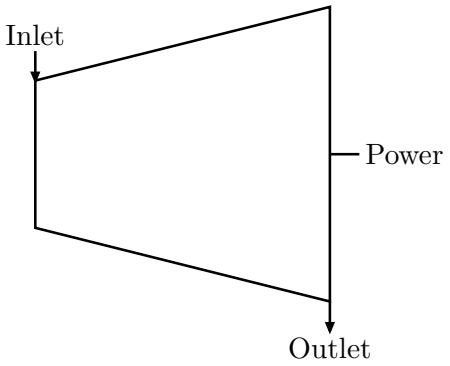

Figure 1: Schematic of a turbine

where T is in Kelvin.
This equation has a few exceptions, i.e, it is only defined for $|p_i| > |p_o|$ and positive T. This may be remedied by a few manipulations to obtain:

$$C_T^2(p_i^2 - p_o^2) - \dot{m}^2 T = 0 \tag{2}$$

However, this formulation does not ensure a physically correct solution, even if the problem converges. The equation has a negative and a positive solution for both $\dot{m}$ and C_T. Physically the negative solution does not make sense, but only by introducing a formulation like:

$$|C_T|C_T(p_i^2 - p_o^2) - |\dot{m}|\dot{m}T = 0 \tag{3}$$

a correct solution is ensured.

3.2 Heat Exchanger Effectiveness

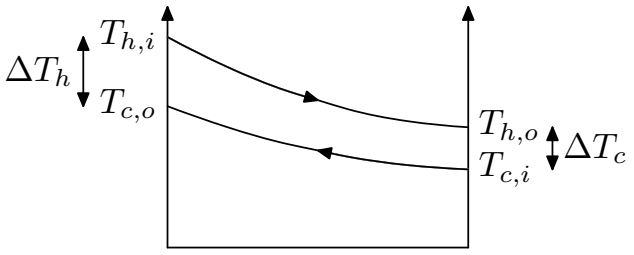

Figure 2: Temperature profile of a heat exchanger

Heat transfer in heat exchangers is driven by a temperature difference. This means that the temperature of one fluid will be higher than that of the other over the whole device as depicted in Figure 2. It may be modelled in several ways. Two common ways are: By introducing an effectiveness as discussed in this section, or by the heat transfer coefficient, discussed next. The two methods are related and models the same characteristic of the device, i.e., how much energy is transferred from the hot fluid to the cold in a double pipe

heat exchanger. In this case, a counterflow exchanger is considered. The heat transfer in this is:

$$\dot{Q} = UA\Delta T_{lm} \tag{4}$$

U is calculated from the flow conditions on both sides of the transfer surface, A. The ΔT_{lm} is an average driving temperature difference for the whole heat exchanger. The effectiveness, ε of the heat tranfer is defined by:

$$\dot{Q} = \varepsilon \dot{Q}_{max} \tag{5}$$

where the maximum possible heat transfer is what would be obtained if one stream exited at a temperature equal to the inlet temperature of the other. The effectiveness is a (non-linear) function of the *Number of Transfer Units, NTU*, which defined as:

$$NTU = \frac{UA}{(\dot{m}c_p)_{min}} \tag{6}$$

For a counterflow heat exchanger the effectiveness relation can be expressed as:

$$\varepsilon(T_{h,i} - T_{c,i}) = \begin{cases} \Delta T_h, \text{ for } (\dot{m}c_p)_h < (\dot{m}c_p)_c \\ \Delta T_c, \text{ for } (\dot{m}c_p)_h \geq (\dot{m}c_p)_c \end{cases} \tag{7}$$

The calculation of the average specific heat, c_p, of a fluid may be approximated by[2]

$$c_p = \frac{\Delta h}{\Delta T} \tag{8}$$

In order to avoid division by zero, the temperature difference should not be evaluated when the current guesses during iteration of the two temperatures of one of the fluids are equal.
Thus algorithm 2 should be introduced to make the model catch this exception.

Algorithm 2 The heat exchanger effectiveness

if $\varepsilon = 0$ **then** {No heat transfer}
  $T_{h,i} - T_{h,o} = 0$
else if $|T_{h,i} - T_{h,o}| < \delta \vee |T_{c,i} - T_{c,o}| < \delta$ **then** {Catching the exception}
  $T_{h,o} - T_{c,i} - \delta = 0$
else
  Use equation 7
end if

This algorithm has been implemented in DNA and converges for any set of guesses of temperatures.

[2]The specific heat may also be calculated in property routines

3.3 Heat Transfer Coefficient

For a counterflow heat exchanger the log mean temperature difference in equation 4 is expressed as:

$$\Delta T_{lm} = \frac{(T_{h,i} - T_{c,o}) - (T_{h,o} - T_{c,i})}{\ln\left(\frac{T_{h,i} - T_{c,o}}{T_{h,o} - T_{c,i}}\right)} \tag{9}$$

This expression is more complicated to handle as the logarithm in the denominator has several exceptions. It was found to converge for many sets of temperatures if it is implemented as presented in [6] and has been improved for this study, see algorithm 3.

Algorithm 3 Heat transfer coefficient model

if $T_{h,i} = T_{c,i}$ **then** {The inlet temperatures are equal}
  $h_{h,i} - h_{h,o} - \delta = 0$
else if $T_{h,o} > T_{h,i}$ **then** {Positive temperature difference of hot fluid}
  $T_{h,o} - T_{c,i} - \delta = 0$
else if $T_{c,i} > T_{c,o}$ **then** {Negative temperature difference of cold fluid}
  $T_{h,i} - T_{c,o} - \delta = 0$
else if $T_{c,i} > T_{h,o} \wedge T_{c,o} > T_{h,i} \wedge UA > UA_{lim}$ **then** {Negative temperature difference at both ends and high UA}
  $h_{h,i} - h_{h,o} - \delta = 0$
else if $T_{c,i} > T_{h,o} \wedge UA > UA_{lim}$ **then** {Negative temperature difference at cold end and high UA}
  $T_{h,o} - T_{c,i} - \delta = 0$
else if $T_{c,o} > T_{h,i} \wedge UA > UA_{lim}$ **then** {Negative temperature difference at hot end and high UA}
  $T_{h,i} - T_{c,o} - \delta = 0$
else if $(T_{c,o} > T_{h,i} \vee T_{c,i} > T_{h,o}) \wedge (\dot{m}c_p)_h > (\dot{m}c_p)_c$ **then** {Negative temperature difference at any end and highest capacity of hot fluid}
  $T_{h,i} - T_{c,o} - \delta = 0$
else if $(T_{c,o} > T_{h,i} \vee T_{c,i} > T_{h,o}) \wedge (\dot{m}c_p)_h \leq (\dot{m}c_p)_c$ **then** {Negative temperature difference at any end and highest capacity of cold fluid}
  $T_{h,o} - T_{c,i} - \delta = 0$
else if $T_{h,i} - T_{c,o} = T_{h,o} - T_{c,i}$ **then** {Same temperature difference at both ends}
  $UA(T_{h,i} - T_{c,o}) - \dot{m}_h(h_{h,i} - h_{h,o})$
else {No exceptions to catch}
  Calculate the residual according to equations 4 and 9
end if

UA_{lim} is a value of the heat transfer coefficient and area which will result in a very low temperature difference ($\approx 10^{-5}$°C) at one end.

This algorithm has proven very robust in use in DNA both for single components and in system models, e.g., heat exchanger networks. Usually, DNA will provide the model with good guesses for the variables in the system, but the user has to decide a few guesses and has the freedom to insert additional guesses. The latter possibility has been used to test the robustness of this implementation for *UA*-values in the range of 0 to 10000 and with different mass flows and inlet temperatures and guesses of outlet temperatures far away from and both below and above the correct value (±100-200°C). The algorithm finds the correct solutions in all of the tested cases, but the number of iterations may vary much.

3.4 Gasifier

The final example of energy system models which may be improved with regards to robustness is a model of a gasifier. A gasifier converts a solid fuel into a combustible gas and a remaining ash fraction, by adding heat and gasification agent, e.g., steam and/or air, to the fuel. Often the model of such a component is to some extent based on the calculation of chemical equilibrium in the outlet gas, as would be obtained if the reaction "had time enough". The calculation of chemical equilibrium may be carried out be minimizing Gibbs' free energy in the gas summing over all, *k*, constituents of the gas.

$$G = \sum_{i=1}^{k} \dot{n}_i \left(g_i^0 + RT \ln y_i \right) \tag{10}$$

The minimization is constrained because an atom balance for each element (e.g., Carbon, Oxygen, Hydrogen) in the reactants must be made:

$$\sum_{i=1}^{k} \dot{n}_{i,in} B_{ij} = \sum_{i=1}^{k} \dot{n}_{i,out} B_{ij} \tag{11}$$

where index *i*, and *j*, refer to a gas compound and an element, respectively.
The minimization is in DNA done by introduction of Lagrange multipliers[5]. However, in order to be generally applicable the gasifier model must account for all atoms present in usual gasifier product gases. These are Carbon, Oxygen, Hydrogen, Nitrogen, Sulphur, Argon. In some cases the modeller will neglect the minor components and the atom balance for these loses relevance. But, the gasifier model has to be prepared for any of these as the amount of one may vary between being present and zero during a simulation, due to guesses and other components. If an element is not present in the gas the Lagrange multiplier for this constraint will not have any relevance and will result in a singular system of equations.
One way which has been found robust for this problem is to substitute the atom balance for an unpresent element by a simple assignment of a value to the multiplier as in algorithm 4.

Algorithm 4 Gasifier model

if The element is present **then**
  Calculate the atom balance, equation 11
else
  Set the Lagrange multiplier for this constraint to 0
end if

4 Discussion

Above it has been demontrated that even some of the well-known troublesome models ín energy system modelling may be made robust by the proposed idea of iterating in expressions which make the variables enter the feasible region, if this has been left during iterations. It is worth noting that the branches introduced in the model outside the feasible region are much simpler (more linear) expressions than the original models. This means that only few iterations are required to enter the feasible region. However, as this is a remedy it does require more iterations and longer calculation times when the model exits the feasible region during iterations. The number of iterations required and the speed of the solution may be dependent on the actual implementation of the numerical methods and the complete system of equations. It should be mentioned that for some sets of guesses the model of the heat transfer coefficient-method only converges if the parameters of damping and the Jacobian updates of the Newton method is within a given interval. Another complication of energy system models is that they have to work in connection with models of the properties of the substances involved in the system. These usually also involve their own numerical methods (it has been stated that as much as 75% of the calculation time is used in these[15]) and may have exceptions, that can cause trouble. Some variables which my cause trouble are pressure and molar fractions which must be positive. From experience it does not seem to be enough to limit these values to ensure convergence.

5 Conclusion

An idea of ensuring convergence of numerical models of physics by catching exceptions to the equations and branching these to iterate back to the feasible region of the values of the variables has been presented and demonstrated by the application to energy system models. A simple application to the natural logarithm has been implemented and demonstrated in EES. Applications of the approach to models of components in energy systems: turbine, heat exchanger, and gasifier, have been implemented and demonstrated in DNA. It has been shown that these extends the convergence interval of these models to values far away from the feasible region. It is also the experience from actual work with these models in system models.

References

[1] Eugene Allgower and Kurt Georg. Simplicial and continuation methods for approximating fixed points and solutions to systems of equations. *SIAM Review*, 22(1):28–85, jan 1980.

[2] Mohsen Assadi, Per Rosén, and Niklas Ågren. Utvärdering av olika värmebalansprogram, (comparison of different heat balance programs). Technical Report LUTMDN/TMVK–3173–SE, Department of Heat and Power Engineering, Lund Institute of Technology, Sweden, 1995.

[3] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, 1983.

[4] Brian Elmegaard. DNA home page. Internet: `http://www.et.dtu.dk/software/dna`, March 2003.

[5] Brian Elmegaard. *Simulation of Boiler Dynamics – Development, Evaluation and Application of General Energy system Simulation Tool*. PhD thesis, Technical University of Denmark, 1999.

[6] Brian Elmegaard and Niels Houbak. Robust implementation of process simulators and their associated models. In G. Hirs, editor, *Proceedings of ECOS 2000*, pages 325–332, 2000.

[7] Brian Elmegaard and Niels Houbak. Software for the simulation of power plant processes part B – description and application. In *Proceedings of ECOS 2002*, 2002.

[8] Free Software Foundation. Home page of the organization. Internet: `http://www.gnu.org`, May 2003.

[9] I. Giglmayr, M. Nixdorf, and M. Pogoreutz. Vergleich von software zur thermodynamischen proceßrechnung. Forschungsvorhaben 177, VGB, Technische Universität Graz, Technische Universität München, March 2000.

[10] Interduct. Software tools for process integration.

http://www.interduct.tudelft.nl/PItools, 1999. Interduct, Delft University of Technology.

[11] Arne Jakobsen. Considerations about the modelling process. In Niels Houbak, editor, *Proceedings from the SIMS'95, 37th Simulation Conference Simulation in Theory and Practice*, pages 13–20. SIMS Scandinavian Simulation Society, 1995.

[12] S. A. Klein and F. L. Alvarado. *EES, Engineering Equation Solver,* User's Manual. F-Chart Software, 1998.

[13] B. Lorentzen. *Power Plant Simulation*. PhD thesis, Technical University of Denmark, Laboratory for Energetics, 1995.

[14] W. Morton and C. Collingwood. An equation analyser for process models. *Computers and Chemical Engineering*, 22(4/5):571–585, 1998.

[15] J. L. Robertson. The ideal process simulator. *Chemical Engineering Progress*, 85(10):62–66, 1989.

[16] Mordechai Shacham and Neima Brauner. Numerical solution of non-linear algebraic equations with discontinuities. *Computers and Chemical Engineering*, 26:1449–1457, 2002.

A The software DNA

DNA is an energy system simulation tool which has both steady state and dynamic simulation features and has proven useful through several research and student projects involving simulation of e.g., steam power, gas turbines, fuel drying, pyrolysis and gasification, fuel cells, and heat exchanger networks.
Recently, DNA has been introduced in the regular education of energy engineers at the Technical University

of Denmark for teaching energy systems with good results. The experience is that it takes about a week for new users to learn to use the program.
DNA includes

- a modified Newton method solver for steady state models, with the linear algebra based on a sparse matrix technique. The solver is modified so it uses numerically calculated differences instead of differentials and that the Jacobian is only updated "when necessary". The iterations step may be damped as well.
- an up to fourth order, variable step size BDF (Backward Differentiation Formulae) solver for dynamic simulation,
- an extendible component model library which is compiled into the code, and
- routines for calculation of state variable properties, transport properties and radiative properties of fluids and solids, e.g., ideal gas mixtures, water/steam, carbon dioxide and solid fuels and ashes.

DNA is also the name of the modelling language, and the system model written by the user is compiled and simulated in one run by the program also named DNA. DNA is a free software system and may be downloaded via the internet [4]. The source code is also available. DNA is a text-based tool, and at present it does not have a graphical interface. The most commonly used interface to the program is the text editor GNU Emacs. For inclusion of new components in the program a (Fortran) compiler is required. Both Emacs and compilers, i.e., the GNU compiler collection, are distributed as Free software [8]. This means that DNA can be used at no expense.