On the Importance of Scaling in Equation-Based Modelling

Francesco Casella
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Milano, Italy
francesco.casella@polimi.it

Willi Braun
Faculty of Engineering and Mathematics
University of Applied Sciences Bielefeld
Bielefeld, Germany
wbraun@fh-bielefeld.de

ABSTRACT

Equation-based modelling languages adopt a declarative modelling approach, focused on writing the model equations in a clear way and leaving the task of deriving efficient simulation code to the tool. One aspect of declarative modelling is that the use of dimensionally consistent SI units for the physical variables is preferrable; however, in many application areas this can lead to implicit nonlinear systems of equations which are badly scaled from a numerical point of view. This paper shows the negative impact of not dealing with this aspect on a benchmark test case, and then shows how the same performance of manually scaled models can be recovered by suitably exploiting information about the scaling of variables that can be declared by the modeller.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**; *Continuous simulation*; • **Mathematics of computing** → Solvers;

KEYWORDS

Nonlinear equations, Scaling, Equation-based modelling

ACM Reference Format:
Francesco Casella and Willi Braun. 2017. On the Importance of Scaling in Equation-Based Modelling. In *EOOLT'17: 8th International Workshop on Equation-Based Object-Oriented Languages and Tools, December 1, 2017, Wessling, Germany*, Bernhard Bachmann (Ed.). ACM, New York, NY, USA, Article 1, 5 pages.
https://doi.org/10.1145/3158191.3158192

1 INTRODUCTION

Equation-based, object-oriented modelling languages (EOOLs), such as Modelica [5] or gPROMS [1], were introduced at the end of the '90s and are now widely used for system-level dynamic modelling of engineering systems of various nature.

EOOLs follow a declarative approach to modelling, where the focus is put on describing how the physical process works by only writing the process equations, rather than focusing on how these equations will be solved. From this point of view, there is a very strong incentive at always using SI units for all equations and variables, because this ensures equations are always dimensionally consistent, without the need of introducing error-prone and weird-looking numerical factors. If this convention is taken, modelling tools can also perform dimensional consistency analysis and detect modelling errors during the development phase of equation-based component models. For these reasons, always using SI units in physical equations is the standard convention taken by the Modelica community.

The downside of this choice is that extremely badly scaled systems of equations can arise, with potentially dire consequences on the robustness of the numerical solvers employed to solve them, and on their performance in general. For example, the model of a large 1 GW power plant will contain variables related to the mechanical power balance on the generation unit with values around 10^9 W, while the flow coefficient A_v (which is expressed in m^2) of some control valves in the same plant model could be as small as 10^{-4}. The unknown variables of the model will thus span thirteen orders of magnitude, which is dangerously close to the machine-epsilon precision of the floating point representation of real numbers in double precision, about $2 \cdot 10^{-16}$, which is the standard in numerical simulation software.

Another example is the device-level model of a transistors in modern integrated circuits, where currents may be as small as 10^{-9} A, while voltages can have magnitudes around 10 V, thus spanning ten orders of magnitude in the same model.

When these equations are processed by the EOO tool for their solution, it is often the case that implicit nonlinear systems need to be solved, using iterative solvers that are usually based on Newton-Raphson's method. In some cases, e.g., the steady-state initialization of large models, these systems can have significant sizes, possibly well above one thousand equations.

In general, these systems of equations can be expressed in compact form as $f(x) = 0$, where x is the vector of the unknown variables and $f(\cdot)$ is a vector-valued function, each component being the residual of one equation in the system.

Numerical solvers for nonlinear equations rely on several heuristic rules, governing decisions such as how large should be the Δx to compute the system Jacobian $\frac{\partial f}{\partial x}$ by finite differences (if analytical Jacobians are not available), when the Jacobian should be recomputed, and, most importantly, when is the solution accurate enough to stop the iteration. The termination criterion often involves the norm of the residual being less than some threshold, i.e., $||f(x)|| < \epsilon$.

Even though this is usually not declared explicitly in the literature and in the documentation about these solvers, the implicit assumption taken by their developers is that the unknowns of the problem have an order of magnitude close to unity or at least not

EOOLT'17, December 1, 2017, Wessling, Germany

ACM ISBN 978-1-4503-6373-0/17/12…$15.00
https://doi.org/10.1145/3158191.3158192

too far from that, e.g. in the range $10^{-3} - 10^{3}$ so that, e.g., $\Delta x = 10^{-8}$ is a good choice to compute a numerical approximation of a derivative, and $\epsilon = 10^{-8}$ or possibly $\epsilon = 10^{-12}$ are reasonable choices for the iteration termination criterion.

It is apparent from the above-mentioned examples that these assumptions do not hold if the equations of the EOO model are handed over as such to the solver without any scaling process.

The consequences can be potentially fatal: if the termination criterion for the iterative Newton-Raphson solver is $||f(x)|| < 10^{-12}$, one of the equations is a power balance in a 1 GW plant, and SI units are used for all variables, this implies asking for a relative accuracy of the component of $f(x)$ containing the power balance equation of $10^{-12}/10^{9} = 10^{-21}$, which is five orders of magnitude less than the machine-epsilon of double-precision floating point numbers and thus hardly attainable. Conversely, if a current balance in a circuit involving an integrated circuit transistor is solved with a 10^{-12} accuracy, the relative accuracy of the solution will be $10^{-12}/10^{-9} = 10^{-3}$, that is, 0.1%, which would be rather poor.

Unfortunately, for some reason, this topic is considered of little or no academic interest, so there is actually scant literature referring to it. In fact, the authors could only find a couple of references on the related topic of non-linear optimization solvers, for which the numerical solution is even more sensitive to bad scaling than the solution of algebraic equations. In [4], the authors attempted to devise a method for automatic scaling, which however did not provide consistently good performance, while in [2] the authors discussed the impact of correct scaling on the convergence of some nonlinear programming problems. To the authors' knowledge, no references exist on this problem in the context of EOOLTs.

On the other hand, some numerical routines for the numerical solution of nonlinear equations do provide some options for scaling, implicitly pointing out that using this feature could be beneficial for the solver performance. For example, the documentation of the open-source, state-of-the-art Kinsol solver [3] states that "To address the case of ill-conditioned nonlinear systems, kinsol allows prescribing scaling factors both for the solution vector and for the residual vector"; scaled norms using those factors are then employed throughout the solver for various purposes.

The goal of this paper is thus manifold. On one hand, to show that proper scaling is critical for numerical solver perfomance in equation-based models that turn out to be badly scaled because of the use of SI units. On the other hand, to show that the appropriate use of variable scaling information provided by the modeller (e.g. via the nominal attribute, in the case of the Modelica language) can fully address this issue, recovering the same simulation performance of equivalent well-scaled problems. Last, but not least, to raise awareness on this issue in the EOOLT community and promote the study of methodologically sound approaches to solve it.

The paper is structured as follows. Section 2 introduces a benchmark test model and discusses scaling methods. Section 3 reports experimental results obtained with the benchmark model using the Kinsol solver in the OpenModelica tool. Section 4 concludes the paper with final remarks and indications for future research work.

2 METHODOLOGY

The aim of this paper is to answer the two following research questions:

(1) Can bad scaling of variables and equations stemming from the use of SI units be critical for nonlinear equation solver performance in EOO tools?
(2) Is it possible to use scaling information provided by expert model developers when declaring the model variables to recover the same performance that would be obtained with properly scaled (e.g., non-dimensionalized) models?

An equivalent, more abstract formulation of the two research question could be the following:

(3) Can the performance of EOO tools be made invariant to the choice of physical units for its variables, so that SI units can always be used without any performance penalty?

It is apparent that a negative answer to this question would be a serious blow to the credibility of a fully declarative EOO approach to the simulation of physical systems.

2.1 Benchmark test case

In principle, it would be nice to have a large database of real life, complex test cases to assess the impact of bad scaling and to evaluate the performance of scaling strategies. Unfortunately, these types of models tend to be proprietary and/or confidential and usually quite involved to communicate and understand.

To overcome this problem, in this paper we define a simple benchmark test model to assess the performance of an EOOL tool with respect to scaling problems in systems of implicit nonlinear algbraic equations. The benchmark model has the following features:

- it has a simple physical interpretation, that also helps building a mental representation of the problem and understanding how the solution is going to behave;
- it is described by a few equations that can be written down in plain mathematical notation;
- its scaling and consequent ill-conditioning can be arbitrarily changed;
- its number of variables can be arbitrarily changed;
- its nonlinear behaviour can be made more or less severe via some parameeters.

The benchmark model represents the series connections of N pairs of parallel-connected nonlinear elastic elements, as shown in Fig. 1

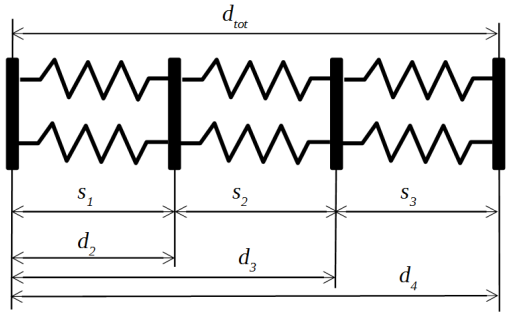

Figure 1: The test case system for $N = 3$.

for the case $N = 3$:

$$d_1 = 0 \tag{1}$$

$$d_{N+1} = d_{tot} \tag{2}$$

$$s_{0,a} = \frac{d_{0,a}}{N} \tag{3}$$

$$s_{0,b} = \frac{d_{0,b}}{N} \tag{4}$$

$$k_i = k_{min} + \frac{k_{max} - k_{min}}{(N-1)}(i-1), \qquad i = 1, \ldots, N \tag{5}$$

$$d_{i+1} = d_i + s_i, \qquad i = 1, \ldots, N \tag{6}$$

$$s_i \left(1 + \left|\frac{s_i}{s_{0,a}}\right|\right) k_i N = F_{i,a} \qquad i = 1, \ldots, N \tag{7}$$

$$s_i \left(1 + \left|\frac{s_i}{s_{0,b}}\right|\right) k_i N = F_{i,b} \qquad i = 1, \ldots, N \tag{8}$$

$$F_{a,i} + F_{b,i} = F_{a,i+1} + F_{b,i+1} \qquad i = 1, \ldots, N-1 \tag{9}$$

where s_i is the relative displacement of the i-th spring pair, d_i is the cumulated displacement of the first i spring pairs, $F_{a,i}$ and $F_{b,i}$ are the forces exerted by the top and bottom spring of the i-th pair, and k_i is the linear stiffness of the springs in the i-th pair.

The linear stiffness coefficients k_i of the springs are uniformly distributed between k_{min} and k_{max} along the system, while the nonlinear behaviour is determined by the parameters $d_{0,a}$ and $d_{0,b}$. If $d_0 \gg d_{tot}$ then the term s_i/s_0 in the spring equations is small and the force-displacement relationship is approximately linear; conversely, the relationship becomes quadratic, which means that the stiffness of the spring increases with its load. The total length of the system is bound to a fixed value d_{tot}.

Note that equations (1)-(5) are explicit and can be solved sequentially, while the rest of the model (6)-(9) constitutes an implicit nonlinear system of equations that needs to be solved simultaneously by an iterative solver.

In the special case $k_{min} = k_{max} = k$, $d_{0,a} \to \infty$, and $d_{0,b} \to \infty$, the model becomes linear and its analytical solution is

$$s_i = \frac{d_{tot}}{N} \qquad i = 1, \ldots, N \tag{10}$$

$$d_i = \frac{i d_{tot}}{N} \qquad i = 1, \ldots, N \tag{11}$$

$$F_{a,i} = F_{b,i} = k d_{tot} \qquad i = 1, \ldots, N. \tag{12}$$

By taking finite values of $d_{0,a}$ and $d_{0,b}$ and $k_{max} > k_{min}$, the problem becomes nonlinear and loses its symmetry, thus becoming more challenging for the nonlinear solver and thus more representative of real-life badly scaled models. On the other hand, if $d_{0,a}$ and $d_{0,b}$ have the same order of magnitude of d_{tot} and k_{max} has the same order of magnitude of k_{min}, one can expect that the order of magnitude of the solution will be the same as that of the ideal solution (10)-(12), which can then be used to estimate sensible scaling values for the problem.

An important property of this model is its scale invariance. It is trivial to prove that if one first solves the problem with certain values of the parameters $k_{min} = k^0_{min}$, $k_{max} = k^0_{max}$ and then solves it again using $k_{min} = \alpha k^0_{min}$, $k_{max} = \alpha k^0_{max}$, α being a positive scale parameter, the solution of the second problem has the same values of s_i and d_i, while $F_{a,i}$ and $F_{b,i}$ get multiplied by α. In other words, scaling the stiffness constants by the same factor α does not change the nature of the problem and its solution, but only changes the scale of the forces by the same amount. In fact, one could interpret α as a unit factor, that could take into account the fact that the forces can be measured with different units, e.g. kN/mm rather than N/m. This property makes it easy to explore the effect of various degrees of ill-conditioning on the solver performance, simply by changing the α factor, without otherwise changing the nature of the system to be solved.

2.2 Scaling Methods

Scaling methods for nonlinear equations stemming from EOO models require three steps:

(1) specifying the order of magnitude of the unknowns;
(2) specifying the order of magnitude of the equation residuals;
(3) solving the equations by taking these factors into account.

The first step conceptually requires the modeller to specify the order of magnitude or scaling value $x_{n,i}$ of each scalar unknown x_i of the problem. In Modelica, this is done by declaring the *nominal* attribute of Real variables.

A convenient way to do this is to declare types with reasonable default values and then use them to actually declare the model variables. For instance, the AbsolutePressure type of the Modelica.SIunits library, which has unit Pa, has a nominal value of 10^6, i.e. 10 bar, which is a reasonable scaling value for most applications not involving extremely high or extremely low pressures. In other cases, the appropriate defaults are more application-specific; for example, currents in modern integrated circuits have order of magnitude 10^{-9}, while currents in typical industrial applications may have orders of magnitude between 1 and 10^3.

The second step can be carried out automatically once nominal values are known, by using the sensitivity information provided by the Jacobian $\frac{\partial f}{\partial x}$. More specifically, the order of magnitude $f_{n,i}$ of the i-th residual can then be estimated as $||v_i||$, where the vector v_i results from the element-wise multiplication of the i-th row of the Jacobian with the vector of nominal values x_n. Either the 1-norm (e.g., the summation of absolute values), the 2-norm (e.g. the Euclidean norm) or the ∞-norm (i.e., the maximum absolute value) can be taken.

The third step depends on the available solver. If the solver has provision for variable and residual scaling, the above-mentioned

Table 1: Solver performance, well-scaled problems

N	k_{min}	k_{max}	$d_{0,a}$	$d_{0,b}$	Iter.	f-eval	Exit
100	1	2	0.5	3	18	47	OK
10	1	2	0.5	3	17	48	OK
1000	1	2	0.5	3	17	46	OK
100	1	2	0.1	1	22	50	OK

information can just be passed to it. If not, the EOO tool should generate the code corresponding to the scaled problem, pass it to the solver, and de-scale the solution as a post-processing step. In other words, defining the scaling factor matrices D_x, D_f and the scaled variables and residuals z and g

$$D_x = \text{diag}\left(\frac{1}{x_{n,1}}, \frac{1}{x_{n,2}}, \cdots, \frac{1}{x_{n,N}}\right) \tag{13}$$

$$D_f = \text{diag}\left(\frac{1}{f_{n,1}}, \frac{1}{f_{n,2}}, \cdots, \frac{1}{f_{n,N}}\right) \tag{14}$$

$$z = D_x x \tag{15}$$

$$g(\cdot) = D_f f(\cdot) \tag{16}$$

the EOO tool can generate the code to compute the residuals (and possibly the Jacobian) for the scaled problem $g(z) = 0$, calls the nonlinear solver routine to solve it, and finally computes the unscaled result as $x = D_x^{-1} z$.

In this context, tolerances and accuracy provided by the end-user as simulation parameters should always be interpreted as relative, non-dimensional values, after the scaling factors have been applied.

3 EXPERIMENTAL RESULTS

In this section, experimental results obtained with the benchmark test model and the Kinsol solver [3] are reported. The OpenModelica platform was employed for the testing. Tearing of nonlinear systems was disabled, to avoid the possibility that all badly-scaled variables could be torn out of the system.

Table 1 reports results obtained with a well-scaled model, which are the reference for solver performance. The values of the model total displacement is fixed to $d_{tot} = 1$, while the other parameters of the model are reported, as well as the number of main iterations of the Kinsol solver, the number of evaluations of the unknown vector, and the outcome of the iterative solution process.

The first line is the baseline case. In the second and in the third, the size of the model is changed, and it seems that this doesn't have a relevant impact on the performance of the solver. In the fourth line, the $d_{0,a}$ and $d_{0,b}$ parameters are reduced, making the model more strongly nonlinear; as a consequence, the number of iterations and of $f(x)$ evaluations increases a bit.

The performance obtained with badly scaled models is reported in Tab. 2, when no scaling is applied and the raw model is directly passed to the Kinsol solver. The first four cases are the same as the first four reported in Tab. 1, except that the spring stiffness coefficients k_{min} and k_{max} were multiplied by a factor $\alpha = 10^9$. In the last two, $\alpha = 10^6$ and $\alpha = 10^{12}$ were taken, respectively. The solver fails in all cases, after a large number of iterations and function evaluations.

Table 2: Solver performance, badly-scaled problems, no scaling

N	k_{min}	k_{max}	$d_{0,a}$	$d_{0,b}$	Iter.	f-eval	Exit
100	10^9	$2 \cdot 10^9$	0.5	3	97	3657	FAIL
10	10^9	$2 \cdot 10^9$	0.5	3	105	4021	FAIL
1000	10^9	$2 \cdot 10^9$	0.5	3	162	6335	FAIL
100	10^9	$2 \cdot 10^9$	0.1	1	553	23017	FAIL
100	10^6	$2 \cdot 10^6$	0.5	3	534	21551	FAIL
100	10^{12}	$2 \cdot 10^{12}$	0.5	3	126	5079	FAIL

Table 3: Solver performance, badly-scaled problems, with scaling

N	k_{min}	k_{max}	$d_{0,a}$	$d_{0,b}$	F_n	Iter.	f-eval	Exit
100	10^9	$2 \cdot 10^9$	0.5	3	10^9	18	47	OK
10	10^9	$2 \cdot 10^9$	0.5	3	10^9	17	48	OK
1000	10^9	$2 \cdot 10^9$	0.5	3	10^9	17	46	OK
100	10^9	$2 \cdot 10^9$	0.1	1	10^9	22	50	OK
100	10^6	$2 \cdot 10^6$	0.5	3	10^6	18	47	OK
100	10^{12}	$2 \cdot 10^{12}$	0.5	3	10^{12}	18	47	OK
100	10^9	$2 \cdot 10^9$	0.5	3	10^6	19	51	OK
100	10^9	$2 \cdot 10^9$	0.5	3	10^4	19	59	OK
100	10^9	$2 \cdot 10^9$	0.5	3	10^2	17	57	OK
100	10^9	$2 \cdot 10^9$	0.5	3	10^0	15	64	OK

Finally, the performance obtained after introducing nominal values for the forces F_a and F_b, as well as the automatic scaling of the problem described in the previous section, are reported in Tab. 3. This table also reports the nominal values used to scale the force variables, reported as F_n. For all other variables, the nominal value was assumed to be one.

The first six cases are the same as the first six cases reported in Table 1, except for the values of the stiffness constants, which are $\alpha = 10^9$ times larger; the correct corresponding nominal value of 10^9 was selected for the force variables. The result, as expected, is that the performance obtained with the well-scaled model is fully recovered.

In the last four cases, an underestimated nominal value was selected for the force variables, three, five, seven and nine order of magnitude smaller, respectively. By comparing with the baseline case reported in the first row, an increase in the number of $f(x)$ evaluation is clearly visible, though the performance degradation is not dramatic.

The last case, in particular, corresponds to a case where all nominal values are taken equal to one. The reason why the performance is still much better than what is reported in the first line of Tab. 2 when no scaling at all is applied, is that in this case the correct residual scaling happens to be applied to the nonlinear spring equations (7)-(8), while the residuals of equations (9) are not correctly scaled. As the latter equations are linear, it seems that they pose less problems during the solver iteration even if not scaled correctly, probably because their residuals become very small after the first iteration. However, it is not always true that equations

involving large-valued variables are linear, so this behaviour is in all likelihood model-specific and not general.

The first important conclusion that can be drawn by comparing Tab. 1 and Tab. 2 is that ignoring the scaling issue completely in the EOO model and in the EOO solver can lead to unacceptable solver performance.

Upon getting this performance from an EOO tool, the end-user may then be tempted to improve the situation by introducing explicit scaling factors in the equation-based model, e.g., by explicitly defining normalized variables and then using them to write normalized equations. However, this would not be the right solution to the problem, because it would be completely against the basic principle of declarative modelling, i.e., decoupling modelling issues from solver issues. In fact, the model would unnecessarily get more cumbersome, error-prone and difficult to read, for reasons which are exclusively related to the numerical performance of the solver.

The second conclusion can be drawn by comparing Tab. 1 with Tab. 3, noting that the correct application of scaling methods allows to recover the same performance obtained with the well-scaled model also with the one which results to be badly scaled due to the consistent use of SI units.

It is worth stressing that in this case the principles of declarative modelling are applied correctly; the modeller is only required to declare reasonable nominal values for the unknowns (which is part of the prior knowledge about the problem), not to explicitly scale the variables nor the equations. The latter task is performed automatically and transparently by the EOO tool, as it is the case for all other equation manipulations that are performed to solve the declarative model efficiently and robustly.

Finally, for the benchmark cases presented here, one may conclude that residual scaling is more important than unknown variable scaling. However, the authors believe that this is due to the nature of the equations whose scaling is not correct in this case, namely Eq. (9). Further investigation, using other test models where equations that only involve large-valued variables are also nonlinear, should be carried out to prove or disprove this claim.

4 CONCLUSIONS

EOO models using SI variables can result in nonlinear systems of equations having very bad variable and residual scaling. Results obtained on a benchmark test case using a state-of-the-art solver reveal that not taking this into account can completely jeopardize the performance of the solver, compared to the case in which equivalent models written with properly scaled variables and residuals are simulated.

We demonstrated that the same numerical performance that is obtained with well-scaled models can be recovered if the modeller provides appropriate nominal values for the badly-scaled unknowns, and if the EOO tool performs suitable variable and residual scaling, starting from that information. This approach allows modellers to safely follow a fully declarative modelling approach, using consistent SI units without worries about potential numerical problems caused by bad scaling.

Further work is required to make this approach fully applicable in an object-oriented modelling context, where models containing appropriate scaling information could be connected to models taken from reusable libraries that do not have this provision. For example, the model of Fig. 1 could be built in a modular way by connecting customized nonlinear spring models, that declare appropriate nominal attributes for the force variables, with standard fixed-point models taken from the Modelica Standard Library, that do not have this provision for the forces declared therein.

In this case, it would be good if the scaling information provided by the custom spring model could be automatically propagated to the fixed point model, avoiding the need of modifying the standard fixed-point model to take into account scaling issues, as well as the need to explicitly provide scaling values as parameters of the fixed-point model.

Suitable methods for automatic propagation of scaling information are thus required, to minimize the amount of scaling parameters to be provided explicitly by the modeller, as well as to trigger warnings when a system model turns out to have incomplete scaling information on variables that end up being the unknowns of implicit nonlinear systems of equations. This could be the subject of interesting future research, combining structural analysis with numerical sensitivity analysis.

REFERENCES

[1] P. I. Barton and C. C. Pantelides. 1994. The modeling of combined continuous and discrete processes. *AIChE Journal* 40 (1994), 966–979.
[2] R. S. Gajulapalli and L. S. Lasdon. 2006. Scaling Sparse Constrained Nonlinear Problems for Iterative Solvers. (2006).
[3] A. C. Hindmarsh, R. Serban, and C. S. Woodward. 2016. User Documentation for Kinsol v2.9.0. (2016).
[4] L.S. Lasdon and P.O. Beck. 1981. Scaling nonlinear programs. *Operations Research Letters* 1, 1 (1981), 6–9.
[5] S. E. Mattsson, H. Elmqvist, and M. Otter. 1998. Physical system modeling with Modelica. *Control Engineering Practice* 6, 4 (1998), 501–510.