# Selection of Numerical Methods in Specific Simulation Applications

Benno Stein and Daniel Curatolo

Department of Mathematics and Computer Science—Knowledge-based Systems
University of Paderborn, D–33095 Paderborn, Germany
stein@uni-paderborn.de

**Abstract.** When designing a technical system, simulation is an important concept to study the behavior of the planned system. Often a cycle of parameter variation and simulation is necessary to analyze the system behavior in detail or to improve the design of the system. Thus, the efficiency by which simulation can be carried out plays a role with respect to time, quality, and cost of the design process.
The paper in hand shows in which way the simulation of technical systems can be speeded up. Starting point is the observation that for the different mathematical problems, which must be solved when simulating a system, several numerical methods are at hand. For instance, a system of ordinary differential equations can be solved by means of an explicit or an implicit Runge Kutta procedure.
Since the different numerical methods are designed with respect to different qualities of a mathematical problem, there exists among the set of competing methods usually one suited best to do the required job. I. e., the qualities of a concrete mathematical problem can be used to select the best method. A central contribution of this paper is to show how this selection process can be operationalized.

## 1 Introduction

During the design phase of a technical system, simulation is an important concept to study the behavior of the planned system. E. g., in our working group we have been developing concepts and tools to support the design of fluidic systems, and we learned about the role of simulation especially within the hydraulic and pneumatic domain. Here, but also in other domains, a cycle of parameter variation and simulation is necessary to analyze the system behavior in detail or to improve the design of the system [4], [13]. Hence the efficiency by which simulation can be carried out plays a role with respect to time, quality, and cost of the design process.

Of course there are other jobs in the course of the design phase that may be much more time-consuming such as the model validation or, in particular, the model formation of a technical system [5]. However, in this place we will focus

on the time factor "simulation efficiency"; the paper in hand shows in which way the simulation of a technical system can be speeded up.

Starting point is the observation that for the different mathematical problems, which must be solved when simulating a system, several numerical methods are at hand. For instance, a system of ordinary differential equations can be solved by means of an explicit or an implicit Runge Kutta procedure.

Clearly, among each set of competing numerical methods, there is usually one method suited best to do the required job. This results from the fact that different numerical methods are designed with respect to different qualities of a concrete mathematical problem. For example, when given a linear equation system in the form $Ax = b$, then the character of the primary diagonal of $A$ along with some convergence criterion decides if to whether a direct method is superior to an iterative method or vice versa.

The simulation of a system, i.e., the processing of its underlying mathematical model, must be performed in a smart manner to gain maximum simulation efficiency. The term "smart" means, that a thorough comparison of a mathematical problem's qualities to the strengths of the existing methods must precede the application of a method to that problem.

Such a postulation raises several questions—among others the following:

1. Which are suited properties to evaluate a particular class of mathematical problems?
2. Given a mathematical problem, which numerical method copes best with this problem?
3. How can the process of selecting a numerical method be automated?

Note that the last question is of interest within two respects. First, tackling the method selection job cannot be expected of the designer of a system, who is challenged getting the knack of the model formation problem at all.

Second, especially in connection with fluidic engineering there exist design tools that automate the model formation step—more precisely—that automatically generate and process a mathematical description when given a CAD drawing as input [7], [9], [14]. Such an automation concept would lose a lot of its charm and productivity if it left the numerical method selection up to the user of the tool.

This paper is organized as follows. The next section conveys an idea of simulation in the fluidic domain: Mathematical problems that occur when simulating a fluidic system are listed along with the necessary numerical methods to solve them.

Given an instance of a mathematical problem $p$ we would like to know which method shall be selected to solve $p$. Answering this question is the purpose of section 3, which shows how a mapping from mathematical problems onto numerical methods can be operationalized. Section 4 presents an example.

Note that the concepts presented here should not be seen as a generic "method selection recipe". Rather, the particular constraints of both the fluidic domain and our objective provide the prerequisites for such a procedure.

## 2 Simulation in the Fluidic Domain

Fluidic systems consist of valves, pipes, cylinders, pumps, etc. The behavior of these components can be described at different levels of detail. Aside from qualitative descriptions, which play a major role in diagnosis tasks, component behavior is quantitatively defined, by equations for the most part.

In order to simulate a fluidic system, which is specified as a mental model e. g. in the form of a drawing, a related mathematical model must be constructed.[1] This mathematical model is comprised of linear, nonlinear, and differential equation systems. Simulating a fluidic system means to solve the underlying mathematical subproblems, so to speak, the equation systems.

Since we deal with a particular domain, our mathematical subproblems may be of a particular structure as well, and there is the question whether knowledge about this structure can be exploited when processing the subproblems.

To answer this question, knowledge about the structure of the mathematical models must be *quantified.* In the following, the interesting mathematical problems along with both selected characterizing properties and methods solving the problems are itemized (cf. e. g. [6], [12]).

- *Linear Equation Systems.*
  Properties: matrix density, size, strength of the primary diagonal, amount of the elements in the primary diagonal, quality of the start vector
  Methods: Gauß elimination, Gauß-Seidel iteration

- *Nonlinear Equation Systems.*
  Properties: size, average amount of coefficients, proportion of linear terms, proportion of higher order terms, quality of the start vector
  Methods: Newton, fast interpolation
  *Remarks.* Due to domain-specific constraints respecting behavior equations and parameter ranges, many nonlinear equation systems are of a particular structure. Thus, an interpolation method for the solution of nonlinear equation systems can be employed. This method is faster than the Newton procedure in many cases but does not always converge.

- *Differential Equation Systems.*
  Properties: modeling deepness, estimated natural frequencies, basis step width, method order, desired precision
  Methods: polygon line procedure, explicit Runge Kutta, implicit Runge Kutta
  *Remarks.* The property "modeling deepness" quantifies the complexity of the components' behavior descriptions. The deeper a component model is the more physical effects does it consider during simulation. Examples for such effects are pressure rises, leakages, or friction.

[1] The construction of a mathematical model for a fluidic system is a demanding problem on its own. In this place we do not engage in model formation details but start at the point where the model is readily set up.

The above list is not complete with respect to the properties of mathematical problems or methods. Moreover, some of the above properties should be discussed in greater detail, respecting fluid-engineering background, of course.[2] Anyway, the list quantifies the idea of what we are looking for: A selection procedure for mathematical methods.

## 3 Automated Method Selection

Automating method selection means to operationalize a mapping from mathematical problems onto numerical methods. Clearly, properties that characterize a mathematical problem can be identified and computed in a straightforward manner. However, the realization of a mapping from these qualities onto a suited numerical method is difficult for the following reasons:

1. Often, a property's effect on a method cannot exactly be quantified.
2. Many properties interact with respect to their effects.

A way out provides the concept of learning [10]: Given are a set of examples each of which defining both an instance of a mathematical problem and the most efficient method for solving that problem. Objective is the identification of knowledge implicitly encoded within the examples—knowledge, which describes the searched mapping. It is possible to pursue such a learning strategy here since important prerequisites are fulfilled:

- In the fluidic domain, large sets of realistic examples can be generated automatically.
- The learning process can be performed off-line.
- Properties which characterize our mathematical problems are available.

The following example illustrates the outlined situation. Let the mathematical problem be the solution of linear equation systems; one of its characterizing properties is the strength of the primary diagonal. Moreover, two numerical procedures that solve the problem, the Gauß-Seidel iteration and the direct Gauß elimination, stand to reason. The strength of a matrix's primary diagonal is given by the minimum quotient $q_{\min}$ of the diagonal coefficients $a_{i,i}$ and the sum of the other elements $a_{i,j,j\neq i}$ of the corresponding line:

$$q_{\min} = \min_{i\in\{1,\ldots,n\}} \frac{|a_{i,i}|}{\sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}|}$$

Figure 1 shows two diagrams each of which containing a plot relating the solution of a set of 1000 linear equation systems. The rank of the equation sytems' matrices is $n = 100$, their occupation density is 25%, and the coefficients have been generated randomly. The left (right) diagram characterizes the runtime behavior of the Gauß-Seidel (Gauß elimination) method dependent on the quotient $q_{\min}$.

[2] Additional information can be found in [3].

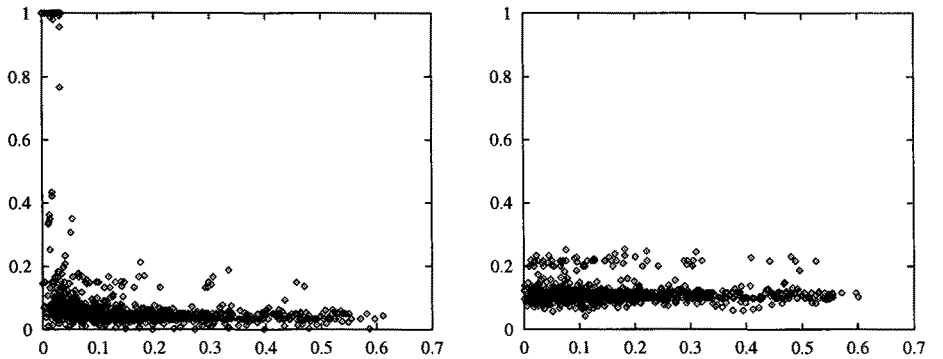


**Figure 1.** The solution time depends on the strength of the primary diagonal. Left diagram: Gauß-Seidel iteration, right diagram: Gauß elimination

Clearly, the diagonal strength has a strong impact on the Gauß-Seidel method, and the diagrams show that equation systems with $q_{\min} \ll 0.1$ should not be solved iteratively. Herewith we got a simple rule that realizes a mapping from a particular mathematical problem onto two numerical methods. It is important to mention that the saved computational effort when choosing the more efficient numerical method exceeds the effort respecting the computation of the decision criterion $q_{\min}$.

The previous example is of a rather simple structure. Of course, many experiments are conceivable where no direct correlation between solution time and problem property can be observed. Instead, a weighted combination of problem properties will be necessary to obtain a meaningful characterization of a mathematical problem.

### 3.1 A Specialized Neural Network for Method Selection

Having compared different learning approaches in the context of our prerequisites, we propose a specialized neural network to realize the learning concept [3]. Here we will not engage into neural network details; foundations and advanced backgrounds may be found in [1], [8], [11], and [2].

The input of the network is formed by numerical values, which define particular properties of a mathematical problem. The network provides as many outputs as there are numerical methods coping with that problem. The network itself is of a rather simple structure, but it comes along with the property that its perceptron neurons can be trained individually. Figure 2 shows the principal structure of the network, which takes values of 4 properties as input and discriminates between three numerical methods.

The network's hidden layer is formed of perceptron neurons, which sum their inputs and compute a value $Meth_{i/j}$ to differentiate between two methods $i$ and $j$. Put another way, the hidden layer's output can be regarded as the runtime benefit of one method over the other. The output layer sums up a method's time benefits. As a result $U_i$ specifies for each method $i$ its appropriateness respecting

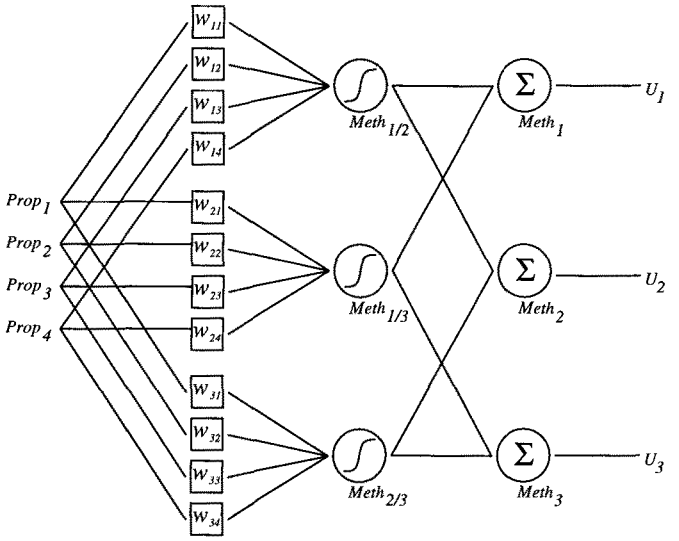


**Figure 2.** Neural network that realizes a mapping for method selection.

the current input vector. The larger the value of $U_i$ the better method $i$ is able in coping with the mathematical problem.

Each perceptron of the network defines a sigmoide transfer function of the following form

$$f(z) = \frac{1}{1 + e^{\alpha + \sum_{i=1}^{k} \omega_i x_i}}$$

where $k + 1$ parameters $\omega_1, \ldots, \omega_k$ and $\alpha$ must be determined by means of a learning procedure. The $\omega_i$ are called synapses weights.

The structure of the neural network was designed having a particular learning procedure in mind. Let us assume that the training data is of the form $(x_1, \ldots, x_n, y)$, where the $x_i$ define the properties of the mathematical problem, while $y$ defines the *runtime difference* of method $i$ over method $j$. Given both samples of that type and an approximation of the samples in the form of a function $f(x_1, \ldots, x_n)$, then $f$ can be used to classify a concrete property vector $(a_1, \ldots, a_n)$ with respect to the two methods $i$ and $j$: If $f(a_1, \ldots, a_n) > 0$, method $i$ is superior method $j$, otherwise method $j$ is superior method $i$.

If $n > 2$ methods for the solution of a mathematical problem stand to reason, the runtime behavior of each method has to be compared to the other $n - 1$ methods, resulting in $\binom{n}{2}$ comparisons. The *absolute* runtime benefit for a method can be computed by summing up its runtime benefits from the $n-1$ comparisons to a total value.

Simplifying matters, this procedure is encoded within the topology of the neural network above. There exist several propositions and strategies elaborating on how the topology of a neural network is to be designed and which learning procedures are adequate. Note that in our case we have a clear idea of how the

net works when it is trained with data of the previously discussed form. Instead of pursuing a global learning strategy, we can make use of this knowledge by training each perceptron of the network individually, by means of regression.

Note that the following important premise must be fulfilled, if a perceptron shall be used for the classification job: There must exist a monotonous connection between the property vector characterizing the mathematical problem and the time difference between the two methods solving that problem.

### 3.2 Determination of the Synapses Weights

By training the perceptrons of our network, the values of the synapses weights $\omega$ are determined. As just outlined, the training can be realized individually for each perceptron here. Aside from efficiency issues or the problem of a learning progress evaluation, the problem of getting stuck in local minima is also avoided by such an approach.

Let us assume that we are given a set of vectors $(x_1, \ldots, x_n, y)$ forming the training data (examples) for a particular perceptron. Then the regression procedure works as follows.

1. $\tilde{y} = \frac{1}{1+e^{\alpha+\sum_{i=1}^{k} \omega_i x_i}}$
2. $\Delta y = y - \tilde{y}$
3. $\Delta z = \beta \Delta y (f^{-1})'(\tilde{y})$, where $f^{-1})'(\tilde{y}) = \frac{1}{\tilde{y}(\tilde{y}-1)}$ and $0 < \beta < 1$.
4. $w_i^{(t+1)} = \omega_i^{(t)} - \Delta z x_i, i = 1, \ldots, k$
5. $\alpha^{(t+1)} = \alpha^{(t)} - \Delta z$

*Remarks.* (*i*) Since the properties of the interesting mathematical problems are of different orders of magnitude, the examples are normalized on an amount $\leq 1$. (*ii*) Within the first iteration, randomized values are used for the $\omega_i$. (*iii*) To avoid oscillation, $\beta$ defines a damping factor, which decreases in the course of the iteration. (*iv*) The adaptation of the weights $\omega_i$ is realized proportionally to the property values $x_i$; the coefficient of $\alpha$ can be set to 1 because of the properties' normalization.

## 4 An Illustrating Example

The presented concepts for the automation of the numerical method selection have been applied to several mathematical problems [3]. This section describes exemplary the case "solution of nonlinear equation systems".

As mentioned in section 2, during the simulation of fluidic systems we fall back onto two competing methods to solve nonlinear equation systems: the Newton method and a fast interpolation method. Each of these methods has its pros and cons, and, given a concrete nonlinear equation system, one method must be chosen.

The following properties of a nonlinear equation system are quantified:

$$\begin{aligned}
\text{size } p_1 &= n \\
\text{average amount of coefficients } p_2 &= \bar{k}_{i,j} \\
\text{number of already known signs of the solution } p_3 &= \frac{k_V}{n} \\
\text{proportion of linear terms } p_4 &= \frac{k_L}{n} \\
\text{proportion of higher order terms } p_5 &= \frac{k_H}{n} \\
\text{proportion of "'mixed"' terms } p_6 &= \frac{k_G}{n} \\
\text{quality of the cached best inverse } p_7 &= |A\tilde{A}^{-1} - E| \\
\text{quality of the start vector } p_8 &= \frac{1}{n}\sum_{i=1}^{n} f_i(z)
\end{aligned}$$

The examples are composed out of $n$ functions of the form

$$f_i(t, x_1, \ldots, x_n) = \sum_{j=1}^{n} k_{i,j} T_{i,j}$$

where the $k_{i,j}$ are real coefficients, and the $T_{i,j}$ are terms of the following polynomial and exponential form respectively, which are typical for real-world applications:

$$T = x_{r_1}^{s_1} \cdots x_{r_p}^{s_p} \quad p \in \{1,2,3,4\}, \quad s \in [0.5;4]$$

$$T = e^{mx_r+b} \quad m, b \in \mathbf{R}$$

Figure 3 shows a diagram depicting several clusters with points and the graph of the regression function $f(z) = \frac{1}{1+e^z}$. This diagram is the result of the training process. The points mark the normalized runtime benefit of the Newton method (y-axis) over the weighted property vector $z = \alpha + \sum_{i=1}^{8} \omega_i x_i$ (x-axis).

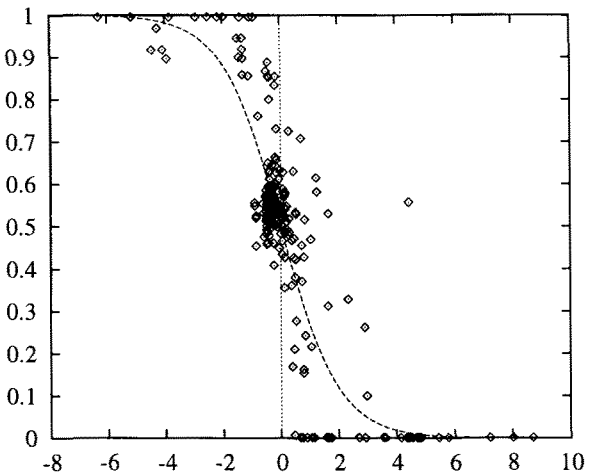


**Figure 3.** Combination of problem properties along with regression function.

The points at the lower right corner characterize large and mid-sized equation systems; here the interpolation method is superior the Newton method.

The cluster of points in the middle characterizes small equation systems; in these cases both methods behave equally efficient. The points in the upper left corner characterize mid-sized equation systems where a good approximation is given because of the existence of the inverse functional matrix from previous computations.

The neural network here has 8 nodes in the input layer (one for each property), a single node in the hidden layer that computes the time benefit between the two methods, and two output nodes. The set of examples consisted of several thousand nonlinear equation systems, which have been derived from 200 hydraulic circuits. Half of the examples were used for training and validation purposes respectively.

The quality of the decision process of the trained neural network is comprised in the following table.

| Optimum decision | Newton method only | Interpolation method only | Neural network decision | % | Example set |
|---|---|---|---|---|---|
| 550 | 3755 | 686 | 593 | **8** | all |
| 210 | 515 | 283 | 229 | **9** | small |
| 340 | 3240 | 403 | 364 | **7** | large |

The first column shows the computation time for the example set, if for each example the best fitting numerical method is chosen. The second and third column show the total computation times when the Newton or the interpolation method are used exclusively. The fourth column shows the computation time if the method selection is controlled by the trained neural network, while the fifth column contains the deviation between the network decision and the optimum decision.

The validation experiments were performed with respect to a differentiation of the examples into three sets, containing small, large, and both types of nonlinear equation systems. The rightmost column reflects this characteristic of the underlying examples.

## 5 Summary

When designing technical systems, aside from model formulation and validation tasks, also the efficiency by which simulation is carried out plays a role with respect to time and quality of the design process.

The paper in hand showed in which way the simulation of technical systems can be speeded up by means of a smart selection of numerical methods for mathematical problems. This selection process was operationalized by means of a neural network.

An interesting particularity here is the topology of the employed neural network, which allows the realization of a local training strategy—more precisely: The training process corresponds to the solution of a regression problem within $n$

dimensions, $n$ denoting the number of qualities investigated for a mathematical problem.

The developed methodology has been applied for simulation problems in the field of fluidic engineering. For the different types of mathematical problems occurring in this field (solution of linear, nonlinear, and differential equation systems, multiplication of matrices) the related network has been instantiated and trained.

Tests with large sets of examples demonstrated the high quality of the decision process that can be realized by our approach.

## References


[1] R. Beale and T. Jackson. *Neural Computing.* Institute of Physics, Bristol and Philadelphia, 1994.

[2] M. E. Cohen and D. L. Hudson. Approaches to the Handling of Fuzzy Input Data in Neural Networks. *IEEE International Conference on Fuzzy Systems*, 1992.

[3] D. Curatolo. *Wissensbasierte Methoden zur effizienten Simulation fluidtechnischer Systeme.* Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 1996.

[4] J. S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.

[5] T. R. Gruber. Model Formulation as a Problem Solving Task: Computer-assisted Engineering Modeling. *International Journal of Intelligent Systems*, 8(1):105–127, 1992.

[6] J. D. Lambert. *Numerical Methods for Ordinary Differential Equation Systems.* John Wiley, New York, 1991.

[7] R. Lemmen. Checking the Static and Dynamic Behaviour of a Hydraulic System. In *Proceedings of the 1th Asian Control Conference, Tokyo*, 1994.

[8] M. Minsky and S. Papert. *Perceptrons.* The MIT Press, Cambridge. Massachusetts, 1969.

[9] Y. Nakashima and T. Baba. OHCS: Hydraulic Circuit Design Assistant. In *First Annual Conference on Innovative Applications of Artificial Intelligence*, pages 225–236, Stanford, 1989.

[10] J. Quinlan. Induction of Decision Trees. *Machine Learing*, 1:81–106, 1986.

[11] R. Rojas. *Theorie der neuronalen Netze.* Springer Lehrmittel Verlag, 1993.

[12] H. R. Schwarz. *Numerische Mathematik.* B. G. Teubner, Stuttgart, 1986.

[13] B. Stein. *Functional Models in Configuration Systems.* Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 1995.

[14] B. Stein and D. Curatolo. Model Formulation and Configuration of Technical Systems. In J. Sauer, A. Günter, and J. Hertzberg, editors, *10. Workshop "Planen und Konfigurieren", Bonn*, volume 3 of *Proceedings in Artificial Intelligence, ISBN 3-92037-97-1*, 1996.