Pontus Boström | Jerker Björkqvist

Optimisation-based black-box testing of assertions in Simulink models

Turku Centre *for* Computer Science

TUCS Technical Report
No 711, September 2005

Optimisation-based black-box testing of assertions in Simulink models

Pontus Boström
Åbo Akademi University, Department of Computer Science
Turku Centre for Computer Science (TUCS)
Lemminkäisenkatu 14 A, 20520 Turku, Finland
Pontus.Bostrom@abo.fi

Jerker Björkqvist
Åbo Akademi University, Department of Computer Science
Turku Centre for Computer Science (TUCS)
Lemminkäisenkatu 14 A, 20520 Turku, Finland
Jerker.Bjorkqvist@abo.fi

TUCS Technical Report
No 711, September 2005

Abstract

Complicated control systems are used in many safety critical applications, such as in cars and airplanes. Due to the nature of these systems verification can be very difficult to do analytically or algorithmically. The only feasible analysis and verification method is often simulation. The generation of good test cases that can expose flaws in models of the system is, therefore, of great importance. In this paper we investigate the use of optimisation methods for finding such test cases automatically. For this purpose we give a language to express assertions in these systems, as well as a translation of the assertions to a form suitable for optimisation. We also discuss different ways to generate the input signals for the system to maximise performance of the optimisation. To evaluate the approach, we provide a case study demonstrating that optimisation methods are beneficial for investigating properties of models of control systems.

Keywords: Control Systems, Simulation, Optimisation, Automated Validation, Testing

TUCS Laboratory
Distributed Systems Design Laboratory
Embedded Systems Laboratory

1 Introduction

Complex control systems are used in everything from home entertainment systems to cars and airplanes. Since control systems are common in safety critical applications their correctness is very important for the safety and reliability. Control systems are often hybrid systems with both continuous components and discrete control logic. In some cases properties about these types of systems can be proved analytically or algorithmically verified. However, many non-linear systems cannot be analyzed in this manner and, therefore, the only feasible verification and analysis method is simulation. Hence, ensuring that a control system functions correctly can be very difficult.

Simulink [10] is a graphical modelling language from MathWorks inc. that has become popular for simulating and analysing dynamic systems. The Simulink framework allows the developer to graphically model dynamic systems using a large library of functional blocks. Models can then be simulated in order to investigate their behaviour. Simulink has been used successfully for developing, among other things, control systems for cars [9, 10] and airplanes [9, 10].

In order to gain confidence that the control system under development functions correctly, we model the system in Simulink and show that certain properties in the model holds. We use assertions to express these properties of the model we want to verify. However, it is not feasible to test the validity of the assertions for all possible inputs and therefore suitable test values have to be chosen carefully. We propose a method for black-box testing of Simulink models where we try to optimise the test input in order to find test cases that expose flaws in the systems. The object function for the optimisation is chosen in such a manner that it approaches the optimum the closer to a false assertion the system is. The optimisation method then enables efficient automatic search for flaws in the system. If no assertion violations are found greater confidence in the validity of the Simulink model can be achieved. To evaluate the performance of the testing method we provide an example and a case study where we investigate how the optimisation method compares to random generation of test cases for finding violations of assertions.

In Section 2 we present related work. Matlab and Simulink are then presented in Section 3. A description of assertions and their translation to a form suitable for optimisation is given in Section 4. In Section 5 the testing methodology is then presented. A small example is given in Section 6 and a larger case study is presented in 7. In Section 8 we conclude.

2 Related work

There are several tools for checking properties of special types of Simulink models. A tool called *Checkmate* [12] can check properties of a special kind of hybrid systems called *threshold event driven hybrid systems*. In such a

system a change in the discrete state can only occur when a continuous variable encounters a threshold. Checkmate provides a library of special blocks and it can check properties about systems modelled using them. A tool has also been developed for translating Discrete-time Simulink models to *Lustre* for verification [1]. Lustre is a formal language with verification support via model checkers.

The Matlab and Simulink framework provides a testing toolbox [9, 10]. This toolbox provides tools for measuring coverage of tests, automatic test generation and tools for managing requirements of the system. It is also possible to insert assertion blocks into Simulink models. The assertions are then checked during simulation. If a false assertion is found the simulation is stopped and the user is notified about the error. However, Simulink does not provide any systematic way to search for assertion violations, which we provide in this paper.

Meta heuristic search is a class of search methods, where the algorithms iteratively search for an optimum of a function that can be non-convex and even non-continuous. *Genetic Algorithms* [3] and *Simulated Annealing* [3] are examples of this type of search algorithms. The first attempt to use Genetic Algorithms for software testing was done by Ellis et al [14]. Their aim was to produce test data for programs written in PASCAL. In his Ph. D thesis [8] Mantere demonstrates the usefulness of Genetic Algorithms for testing various aspects of systems. Among other things black-box testing of systems is discussed. In [13], Tracey et al. present a method for testing that a program complies with a formal specification. The program input is optimised using Simulated Annealing in order to find a input that falsifies the specification. This approach is similar to the approach taken in this paper, but applied to ordinary programs. However, we try to exploit properties of control- and signal processing applications to enable better modelling of properties and increased performance of the search.

3 Simulink

Simulink [10] is a framework provided by MathWorks inc. for the modelling, simulation and analysis of dynamic systems. Dynamic systems can often be described as a system of first-order ordinary differential equations. The system can then be written in the form:

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= g(\mathbf{x}, \mathbf{u}) \end{aligned} \quad , \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

Here the vector $\mathbf{u}$ gives the input signals to the system, $\mathbf{y}$ gives the vector of output signals from the system and the vector $\mathbf{x}$ gives the state of the system.

Simulink consist of a graphical interface where the designer can drag and drop different functional blocks. The blocks are then connected to each other by signals. Simulink have a large library of different blocks modelling

among other things: functions, difference equations, differential equations, data sources and sinks. The blocks can be used together to create systems with both discrete and continuous dynamics. Simulink contains several different differential equation solvers for numerically simulating different types of modelled systems. The model can then be simulated to investigate its behaviour and to check that it satisfies all requirements. Executable code can be automatically generated from the Simulink model as long as the blocks used to model the system belong to the subset of blocks that can be implemented.

Simulink is an integrated part of the Matlab environment. Matlab is a interactive environment for numerical calculations and efficient matrix manipulation. It contains a scripting language where the user can write programs and perform calculations. The simulation of Simulink models can also be completely controlled by scripts from Matlab. Data for simulations of Simulink models can then be read from and written to the Matlab environment. This is used in order to use optimisation routines present in Matlab in our testing method.

4 Assertions

Assertions are boolean conditions often used in software development for stating properties about, e.g., safety or correctness constraints. We like to express assertions over signals. Signals in Simulink can be viewed as total functions from the time interval during which the system was simulated to real values. Hence, the signal y is defined as a total function $y \in [t_0..t_f] \rightarrow \mathbb{R}$, where $t_0, t_f \in \mathbb{R}^+$ and $t_0 \leq t_f$. The assertions describing the properties of the system we like to verify are expressed using first order predicate logic [4]. The language used for constructing the predicates is given below. An assertion Φ is defined as:

$$\Phi ::= \Phi_1 \wedge \Phi_2 | \Phi_1 \vee \Phi_2 | \neg\Phi | \phi(t_c) | \exists t.\phi(t) | \forall t.\phi(t)$$

Here the domain of the time t is assumed to be restricted to the interval $[t_0..t_f]$ over which the system is simulated. The constant t_c is an instance of time in that same interval. The formula $\phi(t)$ is then given as

$$\phi(t) ::= \phi_1(t) \wedge \phi_2(t) | \phi_1(t) \vee \phi_2(t) | \neg\phi(t) | s_1(t) \alpha s_2(t)$$

where s_1 and s_2 are arithmetic expressions on signals and constants. The relation α denotes an relation that can be used to compare arithmetic expressions. Currently the relations $\leq$ and $\geq$ are supported. Equality is not supported since we use floating point numbers in the simulations.

As an example of assertions, consider the two formulas $\Phi_1 = \forall t.(y_1(t) \leq 1 \wedge y_2(t) \leq 0)$ and $\Phi_2 = \exists t.(y_1(t) - y_2(t) \geq 0)$ containing two signals y_1 and y_2. Both the formula Φ_1 stating that y_1 should be less than one and y_2 should be less than zero during the entire simulation, as well as, the formula Φ_2 stating that for some time during the simulation the difference between

y_1 and y_2 should be greater than zero, are valid assertions. Furthermore, the conjunction of the formulas $\Phi_1 \wedge \Phi_2$, disjunction of the formulas $\Phi_1 \vee \Phi_2$ and the negation of the formulas $\neg\Phi_1$ and $\neg\Phi_2$ are valid assertions.

4.1 Real valued representation of assertions

We like to use optimisation techniques to check the validity of assertions. However, boolean expressions are not suitable for optimisation since they can only have the values *true* or *false* and, hence do not provide any direction to search in. In order to make the assertions more suitable for optimisation they are transformed to evaluate to real values. We define that if a predicate θ evaluates to *true* then its corresponding real valued expression θ' is greater or equal to zero and if θ evaluates to *false* then θ' is less than zero.

$$\theta \Leftrightarrow 0 \leq \theta'$$
$$\neg\theta \Leftrightarrow 0 > \theta'$$

Note that the symbol θ denotes either Φ or $\phi(t)$.

We need to translate all logical operators in the predicate to return a real value as a result in order to use them for optimisation. The rules for the operators are chosen in manner that ensures that the smaller the result is the assertion is closer to being *false*. Hence, we get the following rules for using these operators with real valued assertions:

1. $a \leq b \Leftrightarrow b - a \geq 0$
2. $a \geq b \Leftrightarrow a - b \geq 0$
3. $\theta_1 \wedge \theta_2 \Leftrightarrow \min(\theta_1', \theta_2') \geq 0$
4. $\theta_1 \vee \theta_2 \Leftrightarrow \max(\theta_1', \theta_2') \geq 0$
5. $\neg\theta \Leftrightarrow \text{if } (\theta' \geq 0) \text{ then } -(\theta' + 1) \text{ else } -\theta' \geq 0$

Here a and b are arithmetic expressions and θ is a formula of the form Φ or $\phi(t)$. The quantified expressions are also translated. The expression ϕ is a function of time and the universal quantifier corresponds to computing the minimum of $\phi'(t)$ for all t in the interval $[t_0..t_f]$. The existential quantifier corresponds to computing the maximum.

1. $\forall t.\phi(t) \Leftrightarrow \min_t \phi'(t) \geq 0$
2. $\exists t.\phi(t) \Leftrightarrow \max_t \phi'(t) \geq 0$

The rules above are then applied recursively until the entire first order predicate logic formula has been translated. The translated assertions now evaluates to a real value with appropriate properties and can thereby be used for optimisation.

4.2 Checking boolean assertions in dynamic systems

Using the translated assertions we can now state the problem of deciding the validity of an assertion as an optimisation problem. Assume we have a dynamic system as described in Section 3 with input signals $\mathbf{u}(t)$, output signals $\mathbf{y}(t)$ and an assertion to be checked $\Phi(\mathbf{y})$. The assertion is first translated to its real valued representation Φ'. The optimisation problem is then given as:

$$\min_{\mathbf{u}(t)} \Phi'$$

This means that in order to decide if an assertion holds, we need to find the input signal $\mathbf{u}(t)$ that minimises the translated assertion. If the minimum is less than zero a violation of the assertion was found.

In order to handle the quantified expressions $\forall t.\phi(t)$ and $\exists t.\phi(t)$ during the optimisation $\min \phi'(t)$ or $\max \phi'(t)$ needs to be computed. If $\phi(t)$ is continuous this would be very difficult or even impossible. If the system is discrete, i.e. signals are piecewise constant, the expression $\phi(t)$ is given as sequences of values and $\min \phi'(t)$ and $\max \phi'(t)$ can be easily computed. The optimisation problem then becomes to find an optimal sequence $\mathbf{u}(1), \ldots, \mathbf{u}(n)$ of values where the real valued representation of Φ, Φ', is minimal. This problem is well suited for optimisation by existing optimisation methods, since there is a finite number of inputs and the object function Φ' can be easily computed. The approach no longer work for very long sequences or when $\mathbf{u}(t)$ is continuous. We can then use parameterised functions to construct the input signal $\mathbf{u}(t)$.

Note that a simulation in Simulink is always discrete since it uses only numerical differential equation solvers. The object function Φ' can, therefore, always be computed efficiently. However, in general the the number of evaluations and instants of time the system will be evaluated at are not known in advance, which leads to problems if the input is given as a sequence $\mathbf{u}(1), \ldots, \mathbf{u}(n)$ for continuous systems.

The object function Φ' is in the general case not convex and, therefore, there are no numerical optimization methods that are guaranteed to find the optimum [5]. We also restrict ourselves to only check the the system for a finite time interval. Hence, this method for checking assertions cannot guarantee that all possible assertion violations are found. However, optimisation techniques can often perform well on problems like these and give valuable insight in the behaviour of the system.

4.2.1 Continuous systems

When the input to a system is a continuous signal or when the discrete system require very long input sequence we have to represent the signal $u(t)$ using a function $f(\mathbf{w}, t)$. The function $f(\mathbf{w}, t)$ is a function of a vector of parameters $\mathbf{w}$ and the time t. The problem is similar to the problem of function approximation, which has been extensively studied. Often used

approximations are *Fourier series* and *Radial Basis Functions* (RBF) [11]. Fourier series can be described as:

$$f(\mathbf{w}, t) = \sum_n w_n e^{-jn\omega t}$$

The angular frequency ω is given as $\omega = 2\pi f$, where f is the fundamental frequency. This function can written using only sine and cosine functions which, is better suited for our purpose since it does not then contain complex numbers:

$$f(w_0, \mathbf{a}, \mathbf{b}, t) = w_0 + \sum_n (a_n \cos(n\omega t) + b_n \sin(n\omega t))$$

A disadvantage with Fourier series is that the frequency content of the signal is fixed the at the beginning and, hence it cannot be changed by the optimisation procedure. The Radial Basis Functions normally consists of a sum of Gaussian functions:

$$f(\mathbf{w}, \sigma, t) = \sum_n w_n e^{-(t-t_n)^2/2\sigma_n}$$

The parameter w_n gives the height of the Gaussian and σ_n gives the width. Both these parameters can be optimised to obtain the best result. Fourier series are better for representing periodic signals while Radial Basis Functions are better for representing transient signals.

Note that the difference between our problem and function approximation is that we try to find a vector $\mathbf{w}$ that minimizes an assertion while function approximation methods minimizes an error function. Furthermore, we have an additional constraint that $f(\mathbf{w}, t)$ have to be within certain bounds, since signals are usually bounded in real control systems. This leads to an constrained optimisation problem with the constraint $f_{min} \leq f(\mathbf{w}, t) \leq f_{max}$.

5 Testing Simulink models

The primary use of assertions in Simulink is to state model consistency constraints and safety properties. We can state that certain physical laws hold, for example, that energy or momentum is conserved. We can also state assumptions we have made, for example, that pressure is always positive. Safety properties such as the position or pressure should be within certain bounds can also be expressed using assertions. In our framework we can also state some restricted liveness properties. This means that we can state that something will happen eventually, e.g., that the pressure will reach a certain point.

The validity of the assertions is checked by using optimisation techniques. To evaluate the optimisation approach we give two examples where we test different assertions. The first example is a small toy example consisting of a discrete control system. The second example is a realistic model of hydraulic

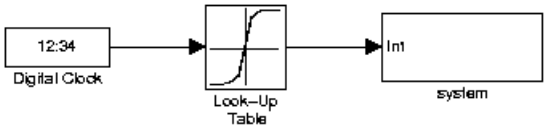

Figure 1: A model for testing a system.

servo system, similar to the model found in a paper by Linjama et al, [7]. All tests in this paper were performed on a PC with Intel Pentium 4 CPU at 2.8GHz and 1GB of memory. We used Matlab version 6.5R13 and Simulink version 5.0R13 running on the Fedora Core 2 Linux distribution. Execution times given in this paper should be considered with this in mind.

5.1 The testing environment

We use optimisation routines implemented in Matlab for checking the assertions. The assertion expressions are also computed using the Matlab scripting language and not modelled in Simulink in order to keep the Simulink models readable. Hence, the Simulink model to test needs to contain blocks for obtaining the input from the optimiser in Matlab and for providing the signal values needed in the assertion expression. The input for the model can be read by a suitable block from the Matlab environment. The values from the model needed for the assertion can be exported to the Matlab environment with *toWorkspace* blocks.

The architecture of a discrete test model is shown in Figure 1. This model is used for discrete input signals where the vector of optimised values is used directly as input sequence to the system. The sub-system block, *System*, is the system to test. The lookup table gives the optimised values via a variable SIMIN. The digital clock gives the time instances when the values are looked up in the table. The object function then has the structure:

```
function y=obj_function(x,...,model,...)
global SIMIN;
SIMIN=x;
...
sim(model,...);
y=assert(a1,...,an);
```

The parameter x is the input signal values computed by the optimisation algorithm and the parameter *model* is the test model to simulate. The variables $a_1, \ldots, a_n$ are variables created by the *toWorkspace* blocks in the model.

In conclusion, there are five steps to perform before starting to test:

1. Construct a test model that contains blocks for providing input. Add *toWorkspace* blocks for exporting the interesting signal values to the Matlab environment.

2. Choose an optimisation algorithm to use.

3. Create an assertion function that returns a value according to the description of assertions in Section 4.

4. In the object function of the optimiser, set the assertion function to be evaluated after the model has been simulated.

5. Choose initial values to start the optimisation from.

Note that using our optimisation approach we can only check one assertion at the time. However, a script in Matlab can then be used to schedule the check of several different assertions to run automatically.

5.2 Optimisation algorithms

There are several classes of optimisation algorithms that each are aimed at different problems. Local optimisation methods are methods that iteratively search for an optimal value based on the gradient in each iteration. These methods cannot optimise functions that are not convex. To remedy this problem so called global optimisation methods have been developed. There are two types of global optimisation techniques deterministic and stochastic. In deterministic methods the search space is deterministically searched for an optimal value, as in for example MCS [6]. In stochastic methods the search space is randomly search based on some heuristics. Popular stochastic methods are Simulated Annealing [3] that mimics the annealing process used in metallurgy and Genetic Algorithms [3] that mimics evolution in biology. We use an function in the Matlab optimisation toolbox based on gradient search.

The optimisation toolbox in Matlab [9] contains a function called *fmincon* for solving constrained, non-linear, multi-variable optimisation problems. This method is a local optimisation method based on gradient search. It uses *Sequential Quadric Programming* (SQP) [5] or a *Reflective Newton Method* [2] depending on the size of the problem. We have chosen this function because it is fast, easy to use and generally produces good results.

5.3 Choosing initial values

Since the optimisation problems here are not convex, the performance of the optimisation algorithms depends heavily on the initial values. There is no correct choice of initial values and usually good ones are found by experimentation. In general good initial values creates an input for which, the output is close to the truth-limit of the assertion. The limitations of local optimisation methods can also be partially remedied by performing the optimisation multiple times starting from different initial values.

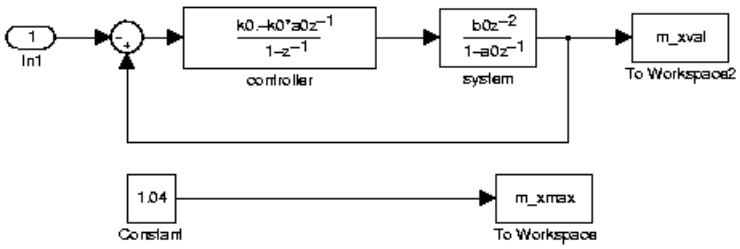

Figure 2: The control system to test.

5.3.1 Random values

The easiest way to choose initial values is to choose random values. If the vector of values is directly used as the input sequence we get a random signal, i.e., a signal consisting of white noise. This type of signal can be useful in some systems but usually it is not a good choice. The vector of random values can also be used for the parameters $\mathbf{w}$ when using a function $f(\mathbf{w}, t)$ to construct the input signal. This also seems to be the best choice for choosing the initial values of $\mathbf{w}$.

5.3.2 A sequence in the form of a step function

Step functions are often used to investigate the behaviour of a system. Here we use a step function as a starting point for the optimisation procedure. The step function can only be used successfully when the vector is used as the input sequence to the system.

5.3.3 Two step optimisation

The optimisation can be performed in two steps. For example, in the first step the frequency of a sine wave can be optimised to discover the frequency for the maximum of the systems frequency response. A sequence consisting of a sine wave of that frequency can then be used as a initial value for the actual optimisation.

6 A small example

To demonstrate our approach to checking assertions we have made a small example. The example consists of a discrete linear system controlled by a PI-controller. The system is shown in Figure 2. The transfer function of the system to be controlled is:

$$G(z) = \frac{b_0 z^{-2}}{1 - a_0 z^{-1}}$$

The parameters a_0 and b_0 have the values 0.8 and 1.4, respectively. The controller is constructed in such a manner that the closed-loop system will

have a overshoot of less than 4% for a step function. The controller is then:

$$C(z) = \frac{k_0 - k_0 a_0 z^{-1}}{1 - z^{-1}}$$

Where the parameter $k_0 = -1/(3b_0)$. We here test if the 4% overshoot limit can be violated with the optimised inputs. The input signal is limited to be in the range [0..1]. The assertion becomes $\forall t.(xmax(t) \geq xval(t))$ where $xmax$ is 1.04 for all t and $xval$ is the output signal of the system.

6.1 Evaluation

In order to test how well the optimisation method work we here compare the optimisation function *fmincon* in the Matlab optimisation toolbox with random search for minimising the assertion. The random search is performed by simulating the system 2000 times with random values. The minimum of the assertion for these simulations is then chosen. As a measure of performance of the method, we test how close to the truth-limit of the assertion the method gets. The closer to the limit the system is, the more stress the method puts on the system. To estimate how consistent the results of the optimisation are depending on initial values, the system is tested ten times for each sequence length and method. The average of the found minimums and standard deviations are then computed.

6.1.1 Direct sequence with random initial values

In the first test of the system, the optimised vector of values was directly used as input sequence to the system. The test was performed with sequences of length 50, 100 and 500 with random initial values. Figure 3 shows the distributions of the minimums found with *fmincon* and random search for sequences with length 100. The results obtained with *fmincon* are concentrated fairly close to 0.0021, while results from random search are more spread out and considerable worse.

The results for all sequences lengths are presented in Table 1. The func-

	minimum	average	standard deviation
fmincon 50	0.0016	0.0023	0.00057
fmincon 100	0.0017	0.0020	0.00039
fmincon 500	0.0017	0.0021	0.00032
random search 50	0.0351	0.0531	0.0104
random search 100	0.0302	0.0514	0.0113
random search 500	0.0354	0.0446	0.0056

Table 1: Minimum with random initial values.

tion *fmincon* finds considerably lower values of the minimum of the assertion

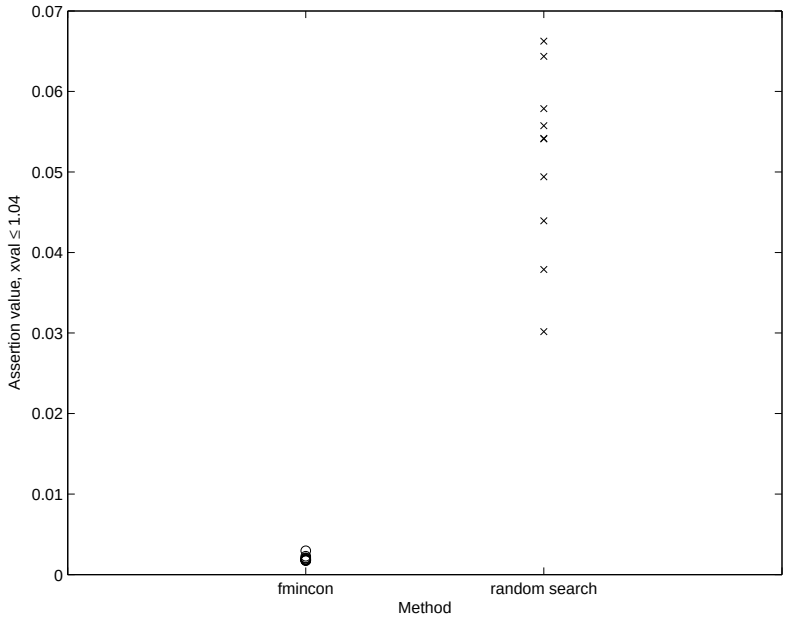

Figure 3: Distribution of the minimums of the assertions for sequences with length 100.

for all sequence lengths. However, the standard deviation is significant meaning that the performance depends on the initial values. This is expected since *fmincon* is not a global optimisation procedure.

The number of times function *fmincon* evaluates the system depends on the size of the input sequence as shown in Table 2. For long sequences

	obj. fun. calls	time in *fmincon*	time in obj. fun.
fmincon 50	750	3.83s	3.29s
fmincon 100	1577	7.92s	6.89s
fmincon 500	7164	53.2s	39.4s

Table 2: Average execution times of *fmincon*.

the object function is evaluated several times more than for random search. The number of evaluations seems to scale linearly with the length of the input sequence. The overhead created by *fmincon* also becomes larger as the length of the sequence increases. It even seems to be increasing more rapidly than the increase in the number of evaluations. The progress of the optimisation of a sequence of length 100 with *fmincon* and random search is shown in Figure 4. The result converges quickly close to the optimal value for *fmincon*. In random search the result is improved in steps when a new optimal value is randomly found.

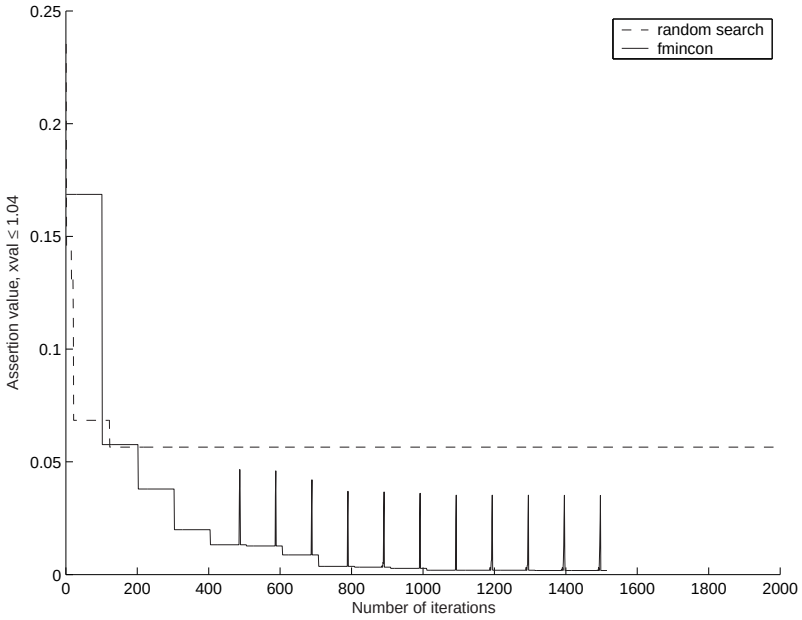

Figure 4: The progress of the optimisation for a sequence of length 100.

	Minimum	Average	Standard dev.
fmincon	0.0400	0.0594	0.0332
Random search	0.0400	0.0400	7.3443e-7

Table 3: The results when using a sine function as input signal.

6.1.2 Step function as initial value

As a second test a step function was used as a start sequence. Again the optimised vector of values was used as the input sequence and the number of elements in the sequence was 100. The step from 0 to 1 was placed after element 50. The step function forces the system fairly close to the truth limit of the assertion. The aim of the test was to investigate if the optimisation procedure could minimise the assertion further. The value of the assertion for a step function was 0.0030 and it could not be improved by the function *fmincon*. The step function seems to be a local optimum and a global optimisation method would be needed.

6.1.3 A sine wave

In the last test the input consisted of a sine wave. The input signal $u(t)$ is described as $u(t) = \sin(2\pi ft)$, where the frequency f is optimised. The initial value for *fmincon* was chosen at random. The results are displayed in Table 3. The optimal value of the assertion that can be achieved with this input function is 0.0400, since the maximum of the frequency response

is one. Both methods find the optimal value. However, random search finds it more often. The number of iterations used by *fmincon* is considerable lower than for random search. The object function is on average evaluated 23 times during a call to *fmincon*, while it is evaluated 2000 times using random search.

6.1.4 Discussion

The function *fmincon* produces significantly better results than random search in the first test. However, it is not a global optimisation procedure and therefore the results depend on the choice of initial values. The length of the sequence also influences the result and it also affects the execution time. The number of times the object function is evaluated for a sequence of 500 elements is almost 10 times the number it is evaluated for a sequence of 50 elements, while the best result is obtained with sequences consisting of 50 elements. Furthermore, the optimisation using *fmincon* performed for a sequence with length 50 evaluates the object function less than half the times random search evaluates it, which leads to a significantly lower execution time. The sine function in the last test cannot produce good results in this example due to its properties.

7 A case study

To evaluate the optimisation based testing method on a larger system we have used our method to check assertions in a model of a hydraulic servo system, similar to the one found in [7]. The model consists of a hydraulics system with controller and model of the physical system. The physical system consists of a load mass and a hydraulic cylinder driven by a servo valve. The aim of the controller is then to control the position of a load mass.

The Simulink model of the system is shown in Figure 5. It is fairly large consisting of more than 200 blocks. The input to the model consists of two signals v_{ref} and x_{ref}. The signal v_{ref} is the desired speed and x_{ref} is the desired position of the load mass. The signals are sampled with a sampling time of 40ms. The sub-system *Closed-loop controller* contains the feedback controller for the system. The pump providing the supply pressure to the system is modelled in the sub-system *Pump*. The sub-system *Valves* models the behaviour of the valves that control the pressure in cylinder. The hydraulic cylinder is modelled by the sub-system *Cylinders* and the load mass is modelled by the sub-system *Mechanism*.

7.1 Testing strategy

The aim of our tests is to find functions for the inputs v_{ref} and x_{ref} that makes the system violate an assertion. The system can use both feed-forward control of speed (v) and position (x) feedback to move from one position to

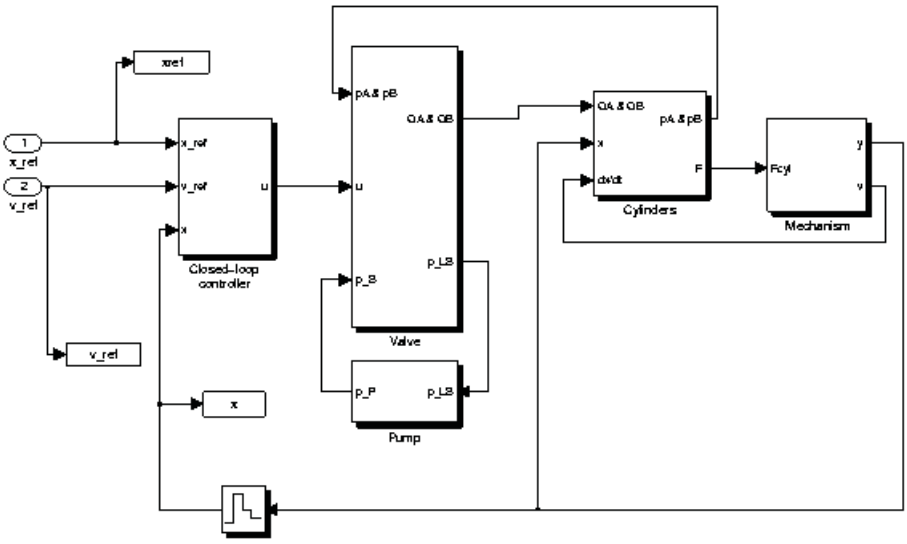

Figure 5: Overview of the model.

another. For simplicity we only use only feedback control of position. This comes at a small cost of slightly degraded performance.

We do tests using two different assertions stating different properties of the system. We compare the function *fmincon* and random search as well as different ways to represent input signals. Both *fmincon* and random search are tested five times for each way of generating input signal and method for choosing initial values. The minimum, average and standard deviation from the five tests are computed to get a better overview of how well each method performs. Due to time considerations and problems with convergence we limit the optimisation algorithms in *fmincon* to evaluate the object function at most 2000 times. The random search procedure again also simulates the model 2000 times and chooses the minimum of the assertion from these simulations.

7.2 The first assertion

The first assertion to test states that the position x should always be smaller than a position x_{max}, $\forall t.(x(t) \leq x_{max})$, where $x_{max} = 0.20$. The range of the input signal x_{ref} is limited to the interval 0..0.15. Two different ways to represent the input signal are used. First the optimised vector is directly used as input sequence and then a sine wave with optimised frequency is used.

7.2.1 Direct sequence and random initial values

First we tested using the vector directly as the input sequence x_{ref} to the system. The initial values were chosen at random. The minimums of the

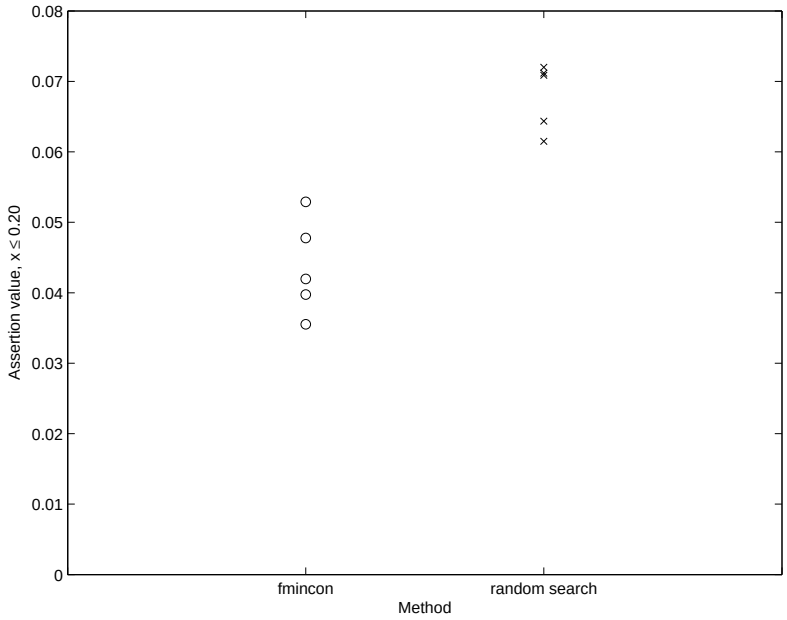

Figure 6: Minimums of the assertion for a sequence of 100 values.

assertion for the optimisation with *fmincon* and random search are shown in Figure 6. As in the simple example in Section 6 the results for *fmincon* are better. However, the variation in the result of *fmincon* is much larger due to the increased complexity of the model.

The average and standard deviation for the obtained minimums of the assertion were also computed as shown in Table 4. The function *fmincon*

	Minimum	Average	Standard dev.
fmincon 100	0.0355	0.0436	0.0068
Random search 100	0.0615	0.0680	0.0047

Table 4: Direct sequence with random initial values.

performs better than random search with both a lower minimum and average of the minimum of the assertion. However, in the corresponding test on the small example in section 6, *fmincon* performed about 20 times better that random search and now it performs only about 2 times better. This is due to the increased complexity of the model. The standard deviation is also high since the optimisation algorithm depends on the initial values.

The execution times become fairly long due to the size of the model and the large number of times the model has to be simulated, as shown in Table 5. The object function is evaluated about 2000 times for each call to *fmincon*. The overhead associated with *fmincon* becomes very small compared to the simulation of the model. The random search is faster probably due to how

	Number of evaluations	Total time
fmincon 100	1	578.5s
Object functin	2018	570.5s
Random search 100	2000	442.0s

Table 5: Execution times for a sequence of length 100.

Matlab handles function calls.

7.2.2 Direct sequence and step function as initial value

As a second test we checked how much closer to the truth-limit the optimisation algorithms get from a step function. Again 100 samples were used and the step was at sample 50. The step function and *fmincon* are deterministic and, hence the optimisation only needs to be performed once. The result

	Minimum
Step function 100	0.0040
fmincon 100	0.0020

Table 6: Sequence of 100 values and a step function as initial value.

of the optimisation is shown in Table 6. The optimisation can improve the result significantly, as opposed to corresponding test on the small example in Section 6. Note that the step function provides a smaller minimum than any of the minimums obtained when starting from a random sequence. The object function is here evaluated 2002 times using *fmincon* and, hence this initial value does not improve the speed of the optimisation procedure.

7.2.3 Sine wave

In this test the frequency f of a sine wave was optimised to minimize the assertion. The formula for the input signal x_{ref} is given as $x_{ref}(t) = 0.075\sin(2\pi ft) + 0.075$. This formula ensures that x_{ref} is also here in the same interval [0..0.15] as in the previous tests. The system was simulated using a time interval of 5 seconds and the initial value was chosen randomly. The result is shown in Table 7. Random search is actually better than *fmin-*

	Minimum	Average	Standard dev.
fmincon 1	0.0211	0.1013	0.0448
Random search 1	0.0105	0.0128	0.0021

Table 7: A sine wave with random initial value.

con in this case. However, the minimum of the assertion is not as good here

as in the previous test. The procedure *fmincon* often get stuck in a local optimum and, therefore it cannot find better values.

The object function is evaluated on average 36 times using *fmincon* compared to 2000 times for random search. This means that the optimisation using *fmincon* can be performed more than 50 times in the same time as one random search. Hence, several optimisations using *fmincon* can be performed using the same mount of time as one random search and thereby improve the result.

7.3 The second assertion

The second assertion states that the position x should never be greater than the reference position x_{ref} plus a certain limit, $\forall t.(x(t) - x_{ref}(t) \leq limit)$. To test this assertion we use Fourier series and the sine wave with optimised frequency to construct the input. Fourier series were not used for testing the previous assertion, since it is difficult to control the precise maximum of the input signal using them. Here the value of the assertion is not directly related to the maximum of the input signal. We do not use the optimised vector directly as input sequence here, since *fmincon* does often not terminate for this assertion. This is due to a bug where *fmincon* fails to terminate on certain ill-conditioned problems. However, the bug is fixed in newer versions of *fmincon*.

When using Fourier series we optimised the value of 11 parameters (1 constant term, 5 sine waves and 5 cosine waves), where each parameter was in the interval 0..0.05. The initial values were chosen at random. The simulation time was 4 seconds and we have that $f = 1/T$ and $T = 4s$. The sine wave was generated in similar manner as previously in Subsection 7.2, $x_{ref}(t) = 0.15\sin(2\pi ft)$. Note that the frequency components of the Fourier series is limited by the relatively small value of the weights w_i, which might give the sine wave an unfair advantage.

The results are shown in Figure 7 and summarized in Table 8. Here the

	Minimum	Average	Standard dev.
Fourier, fmincon	-0.2697	-0.2372	0.0293
Fourier, random search	-0.1897	-0.1887	0.0040
Sine, fmincon	-0.3376	-0.2186	0.0764
Sine, random search	-0.3372	-0.3363	0.00083

Table 8: Results for the Fourier series and sine wave.

assertion does not hold in the model. Both the function *fmincon* and random search discover this fact for each type of input function, but *fmincon* again produces the better (smaller) results. However, for the sine wave *fmincon* produces a better value only in one instance. The use of a sine wave with optimal frequency seems to be the best way to generate inputs with this

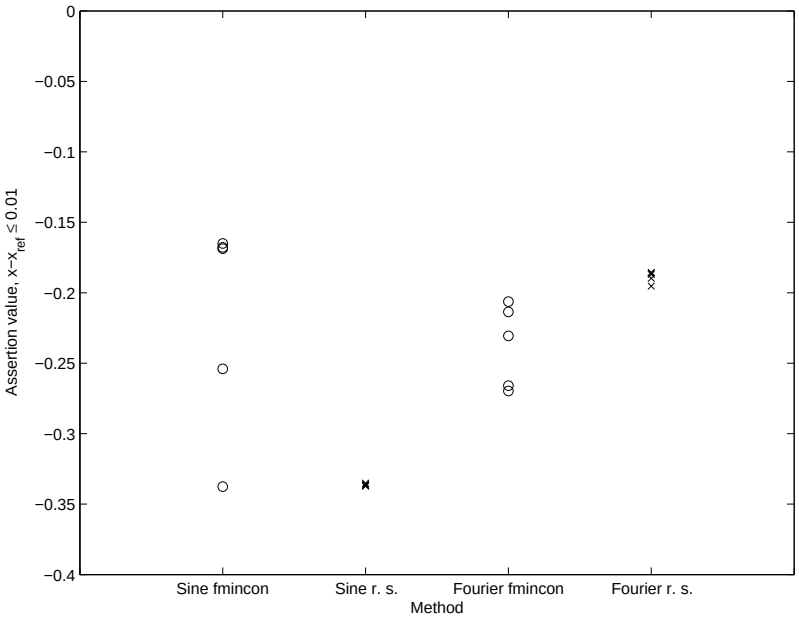

Figure 7: The results obtained when the input signal is a sine wave or a Fourier serie.

assertion. As for earlier experiments the result of the optimisation depends on the initial values.

The advantage of *fmincon* becomes clear when comparing the average number of iterations performed in each test, shown in Figure 8 . When using a sine wave the number of iterations used by random search is more than 50 times the number used by *fmincon*. When using Fourier series almost ten times as many iterations are used by random search. This means that several more tests can be performed using *fmincon* within the time of one random search which, can lead to further improvement in the optimised values.

7.4 Discussion

The system in the case study has non-linear dynamics and is fairly complicated. The optimisation function *fmincon* can still often produce good results but the advantage over random search is smaller than in the small example in Section 6. The choice of input signal representation can often have as much influence on the result as the optimisation method. The choice of initial values is also important. By using a representation of the input signal that requires few optimisation variables the number of iterations required by *fmincon* can be greatly reduced. This is beneficial since, when using a local optimisation method such as *fmincon* the optimisation should be performed several times with different initial values to obtain good results.

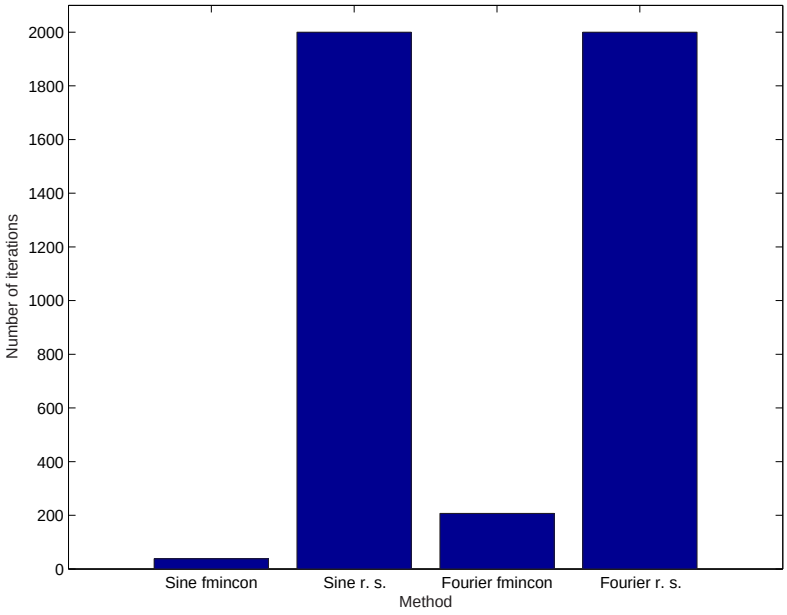

Figure 8: The average number of iterations for each test.

8 Conclusions

In this paper we have formulated the assertion checking problem in Simulink models as an optimisation problem. To enable optimisation based checking of assertions, we have introduced a simple language for expressing the assertions and shown how it is translated to a form suitable for optimisation. We have used a gradient based search method in Matlab optimisation toolbox for checking these translated assertions. This provides a method for automatically generate good test cases for exposing flaws in the system. The optimisation method was compared to random search in a case study and it was shown that the optimisation achieved smaller assertion values, and hence better results, in most cases. The overhead associated with the optimisation algorithm was also shown to be negligible compared to random search for a moderate numbers of optimisation variables.

The optimisation can be further improved by using global optimisation techniques. Randomized methods such as Simulated Annealing [3] or Genetic Algorithms [3] could be very efficient for this purpose. Function approximation methods and technique from the area of adaptive systems and neuro computing [11] could also be investigated for finding optimal parameters to the parameterised functions used to construct the input signals.

Verification of non-linear control system consisting of both continuous and discrete parts is hard. The approach to test case generation presented here provides an automated way to search for good test cases, and that way minimize the risk of errors in the final system. The method cannot guarantee that all assertion violations are found but it can at least increase

the confidence that the system works as intended.

Acknowledgement

The authors like to thank Matti Linjama for providing the model in the hydraulics case study and for the help with using the model.

References

[1] P. Caspi, A. Curic, A. Maignan, C. Sofronis and S. Tripakis. *Translating Discrete-Time Simulink to Lustre*, Technical report, Verimag, France, 2003

[2] T. F. Coleman and Y. Li, An Interior, Trust Region Approach for Nonlinear Minimization Subject to Bounds, *SIAM Journal on Optimization*, Vol. 6, 1996, pp. 418-445,

[3] L. Davies (editor). *Genetic Algorithms and Simulated Annealing.* Pitman, London, 1987.

[4] H. B. Enderton. *A mathematical introduction to logic*, 2:nd edition. Academic Press, 2001.

[5] R. Fletcher. *Practical Methods of Optimization*, John Wiley and Sons, 1987.

[6] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search, J. *Global Optimization*, 14, 1999, pp. 331-355.

[7] M. Linjama, T. Virvalo, J. Gustafsson, J. Lintula, V. Aaltonen and M. Kivikoski. Hardware-in-the-loop environment for servo system controller design, tuning and testing. *Microprocessors and Microsystems*, 24, Elsevier, 2000, pp. 13-21.

[8] T. Mantere. *Automatic Software Testing by Genetic Algorithms.* University of Vaasa, PhD thesis, Finland, 2003.

[9] Mathworks Inc. Matlab. 2005
`http://www.mathworks.com/products/matlab/`

[10] Mathworks Inc. Simulink. 2005
`http://www.mathworks.com/products/simulink/`

[11] J.C. Principe, N. R. Euliano and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals through Simulations*, John Wiley and Sons, 2000

[12] B. I. Silva, K. Richeson, B Krogh, A. Chutinan. Modeling and Verifying Hybrid Dynamic Systems Using Checkmate, In S. Engell, S. Kowalewski and J Zaytoon editors, *proceedings of the 4th international conference on*

automation of mixed processes: hybrid dynamic systems, Shaker Verlag, 2000.

[13] N. Tracey, J. Clark and K. Mander. Automated program flaw finding using simulated annealing. In *International Symposium on Software Testing and Analysis*, ACM/SIGSOFT, 1998, pp. 73-81.

[14] S. Xanthakis, C. Ellis, C. Skourlas, A. LeCall and S. Katsikas. Applying genetic algorithms to software testing. In *proceedings of the 5th International conference on software engineering & it's applications*. Tolouse, France, 1992

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952-12-1607-7
ISSN 1239-1891