System identification using genetic programming

Naus, G.J.L.

Published: 01/01/2003

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. Dec. 2017
System Identification using Genetic Programming

G.J.L. Naus

DCT Report no: 2003.101
November 2003

TU/e Traineeship Report
November 2003

Supervisors:
dr.ir. M.G.J. v.d. Molengraft
ir. D. Bruijnen
Professor:
prof.dr.ir. M. Steinbuch

University of Technology Eindhoven (TU/e)
Department of Mechanical Engineering
Division Dynamical Systems Design
Control System Technology group
System Identification using Genetic Programming

G.J.L. Naus

Professor: prof.dr.ir. M. Steinbuch

Control Systems Technology Group, Department of Mechanical Engineering,
University of Technology Eindhoven

Abstract

GP4SI [4] is a tool for system identification using Genetic Programming. It generates Simulink-models that aim at describing the input-output data of a system. The quality of these models is evaluated by a so-called fitness-function. In this study this fitness-function and the settings of GP4SI are respectively redesigned and optimized. This results in a better understanding of the working and possibilities of GP4SI, guidelines for the settings and a frequency-domain based fitness-function. Several 2nd-, 4th- and higher order example plant-models are successfully identified.

1 Introduction

Genetic programming (GP) [1] is used to design programs that develop on their own with as purpose to improve; the best and most suitable ones will survive, while the others don’t get the chance to reproduce nor to get offspring. The advantage of such a program is that during the optimization and development of the program no input from an operator outside the program is needed.

Bruijnen [3] has examined the possibilities of using GP in designing model-based controllers by means of the system’s input-output data. The first step in his thesis was system identification using GP to obtain a good model as a basis for controller-design. For this purpose he wrote the program Genetic Programming for System Identification (GP4SI) [4]. One of the crucial parts of this program is the so-called fitness-function, which evaluates the models and assigns a fitness-value to each of them. This fitness-value is based on the error between the output data of the system at hand, called the plant, and the model generated by GP4SI, called the GP-model: the smaller the differences, the smaller this fitness-value. So a low fitness-value corresponds to a high quality of the GP-model and assigns a high "mark" to this model. Furthermore GP4SI has a lot of settings and variables to be set that are of great importance to the speed and the accuracy by which a good GP-model is found.

The objective of this internship is bipartite: firstly it consists of the study of the influences of the settings and variables of GP4SI on the speed and accuracy by which a good GP-model is found. Secondly it consists of redesigning the fitness-function. In [3], the fitness-function was formulated in the time-domain, which turned out to be a problem. Therefore, in this work a frequency-domain based approach is followed. Validation of the final settings and the resulting frequency-domain based fitness-function is done by simulations using both Simulink-models and experimentally obtained data.
2 Genetic Programming and GP4SI

2.1 Genetic programming

Genetic Programming [1] is part of the so-called Machine Learning (ML) which is referred to as the automatic evolution of a population of computer programs in such a way that they become more suitable for a given objective. This is done by feeding the program with some input data and comparing the output with the desired behaviour. Koza [2] introduced GP in 1992 and a lot of research on this topic has been done ever since. In his masters thesis Bruijnen [3] examined the possibilities to design a tool for automated controller design using GP by means of the input-output data of a given system [3, 2002-2003]. His thesis was bipartite and consisted firstly of system identification although the main result was the actual controller design. Information about the working of Genetic Programs can be found in Banzhaf [1], Bruijnen [3, sect.2] and the sections 3.3 and 3.4 of this study.

2.2 GP4SI

As a result of the first part of his thesis, Bruijnen wrote the tool Genetic Programming for System Identification (GP4SI) [4]: a tool for automated system identification using GP. It is written in MVC++ 6.0 using the EO-library [12] and further connection is set up with Matlab / Simulink using the so-called Matlab engine: Simulink-models are created, which represent the given plant. The simulations are performed with Matlab and the user has to determine what is good and bad by implementing a fitness-function in an m-file (fitness.m) resulting in the fitness / performance of the Simulink-models.

After loading the settings of gp.ini, GP4SI creates a population of Simulink-models, GP-models, called individuals that are generated randomly and which form together a generation. In these models user-defined Simulink-built-in functions, the so-called nodes, are used. For every model the fitness is evaluated using fitness.m which results in a 'mark' of goodness, the fitness-value, in representing the given input-output data of a plant. If one of the termination criteria (minimum fitness, maximum number of generations or user-intervention) is satisfied, GP4SI stops. Else it creates a new generation by selecting individuals probabilistically according to their fitness-value and applying genetic operations on them. These genetic operations consist of shifting, replacement and removal of individual nodes and branches with nodes and the optimization of the parameters within the nodes. Again, the fitness of the individuals in the new generation is evaluated and the iteration-loop has started. Only the input-output data is needed, which is the major advantage of GP4SI. No other information about the characteristics of the plant has to be available, so GP4SI acts like a black-box. A more detailed explanation of this can be found in Bruijnen

A list of supporting files for GP4SI is included in the users manual (appendix C).
[3] and Banzhaf [1].

3 Settings of GP4SI

3.1 Classes of settings

The way the individuals develop during a GP-run using GP4SI is of a stochastic nature because GP is a stochastic search technique. Its purpose is to find a solution in the complete search space without any impediment of boundary conditions or the like, so it is possible a good solution isn't found after the first run. Therefore GP produces different solutions every run and it is a matter of time and resources for GP4SI to come up with a good solution.

The ultimate result of Bruijnen [3] is a tool, Genetic Programming for Controller Design (GP4CD) for the automated design of controllers. These are model-based controllers so the first step of system identification with GP4SI, which is included in GP4CD, is necessary. During this study, recommendations and settings for GP4SI are developed that are recapitulated in the conclusions and the brief users manual in appendix C. More extensive and general details can be found in the users manual for GP4CD of Bruijnen [3, app. D].

Figure 2.1: schematic representation of the working of GP4SI

The working of GP4SI is based on a great deal of settings and variables defined in gp.ini and fitness.m. A lot of them are standard settings but the others have to be reviewed every simulation. Those defined in fitness.m will be discussed in the next section. The settings of gp.ini can be divided into 5 classes:

1. **GP-settings**: the basic GP-settings;
2. **Genetic operations**: settings that indicate the chances that certain genetic operations are used by GP;
3. **Nodes and node-settings**: selection of the available blocks to build a model and settings for these blocks;
4. **Matlab/Simulink settings**: settings related to Matlab/Simulink;
5. **GA-optimization**: settings for the Genetic Algorithm that performs a global optimization on the parameters of the GP-models.

In earlier versions of GP4SI also a Nelder-Mead Simplex optimization was included that worked simultaneously with the GA-optimization. However it had very little (positive) influence on the outcome and therefore it has been deleted from GP4SI in an early stage already. The 4th class contain merely standard settings, but the 1st, 2nd and 3rd classes involve particularly the settings and variables with substantial influence on the working of GP4SI. The 5th class contains the settings for the GA-optimization,
an important part of GP4SI, but these are already optimized for this particular problem and therefore will not be evaluated in this study. To examine the influence of the other settings, simulations are done while varying per class one or more settings or variables (see app. A and B). In the next subsections the results of these simulations will be discussed, resulting in an interpretation of the influences of the settings of the 1st, 2nd and 3rd class respectively.

3.2 Basic GP-settings

The settings involved in this group influence the basic way of working of the GP-algorithm. First of all in every iteration-step a new generation has to be created: individuals (the parents) are selected to produce offspring. Two often used selection methods are selecting parents probabilistic in proportion to their fitness and selecting the best individual out of a few randomly selected individuals from the population, which is repeated until enough parents are selected. In this study only the latter one called *Tourment selection* is used with 4 individuals that produce offspring each generation, which are chosen out of 2 randomly selected parents. The influences of varying these settings or the use of another selection method is a point for further study.

The *TerminationFitness* is only used with the 2nd-order models (see app. A and B) and the *MaxGenerations* is held large enough to prevent early stops of simulations, which wasn't an issue in this study. The number of *Individuals* is varied according to the success or otherwise of a simulation: if no better model is found at a certain point, one of the possible problems may be the lack of diversity among the present generation. This may be solved by using a larger percentage of mutation or by a greater amount of *individuals* in the generations. This amount is varied between 300 and 800 during the simulations and mentioned with the simulations of table B.5 as it was varied a lot when trying to identify a system of higher order. Generally an amount of about 300 did work appropriately.

The *MaxTreeDepth* indicates the maximum number of successive blocks that may be used in a Simulink-model; the larger the *MaxTreeDepth*, the more complicated the Simulink-models will be generally speaking. So when identifying relatively simple systems, fewer blocks have to be used than when identifying more complicated systems. More complicated models take longer to simulate, so the smaller the *MaxTreeDepth*, the faster the System Identification goes. In this study a *MaxTreeDepth* varying from 4 to 8 is used, dependent on the order of the system.

3.3 Genetic operations

After it has generated a population of models, GP4SI selects a few individuals from this population, the parents, and performs some genetic operations on them. These individuals are chosen probabilistic in proportion to their fitness. The used genetic operations are:

- **Crossover**, which makes a randomly selected branch of each of 2 selected parents exchange. Most of the time it has a negative effect on the fitness of a parent, but it is needed to make huge steps in the search space. It is the most used genetic operation in practice and a usage of about 70% is common;

- **Branch mutation**, which creates a new random branch at a randomly selected node of a tree and therefore assures a high diversity of the population. It is the second most used genetic operation and a usage of about 30% is common;

- **Point mutation**, which works pretty much the same as branchmutation: it places a randomly selected block at a randomly selected node of a tree;
- **GA-optimization**, which optimizes the model parameters using a genetic algorithm;

- **Branch addition**, which was added during this study but is of little importance for system identification (see also section 4).

More information about genetic operations can be found in Banzhaf [1].

The settings of these genetic operations represent the chances that the operations are used by GP4SI; the percentages of usage. These percentages of the various genetic operations to be executed are independent and therefore it is possible that more operations are applied at one tree. The mentioned settings of 70% crossover and 30% mutation are very common, in literature only one different way to apply the settings at a comparable example used by Koza [2] was found. Koza used 95% crossover and 5% for both mutations. The results of the simulations with these settings were very poor (see simulations 5/1-3 of table A.1) and a logical explanation is that there is too little variation, caused by the low mutation chances. In most cases in this study 75% crossover 15% per mutation and 5% branch addition were used and in some cases the mentioned percentages were taken slightly higher to gain more variation.

Little was known about the influences of setting the GA-optimization and therefore it has been varied between 0.0 and 1.0 (times 100% gives the percentage of usage) with intervals of 0.2. At each step various simulations have been performed (see table A.2). From the results the conclusion can be drawn that a model of reasonable quality is found relatively quick when using a small percentage of usage, but that a large percentage of usage results in fitness-values that converge more quickly to their optimal values. With 'optimal values' the minimal fitness of temporary GP-models is meant. The latter, on the other hand takes longer to find a reasonably good model that may give a notion of the final model. So to what extent the GA-optimization is used, influences the development of the simulations and is balancing between the speed of finding and the accuracy of temporary models. If enough time is taken to complete the optimization, the resulting models will be good in either way of course. When looking at figure 3.2 for example, one can see that the development of the fitness in simulations with low percentages of usage behaves something like a \(1/x\) curve and with high percentages of usage more like a \(1 - e^x\) curve. In this case a relatively simple plant is used so relatively few parameters are needed. But with more complex systems more parameters are involved and therefore the parameter-value optimization becomes more important and a higher percentage of usage may be useful. In this study in most cases a percentage of 20% is used.

### 3.4 Nodes and node-settings

By 'nodes' we mean the building blocks that can be used by GP4SI to create Simulink-
Sect. 4: Changes to GP4SI

models. Every block stands for a built-in operation of Simulink except for the divider-block. Because a division by zero may occur, the Simulink built-in divider-function is replaced by a so-called protected division [4]: an S-function that doesn’t return an error in case of a division by zero and therefore prevents simulations to stop early. The available blocks are: add, subtract, multiply, divide, integrate, differentiate, TF 1st-order, TF 2nd-order, gain, constant, input and time-delay. The blocks that are to be used by GP4SI can be set in gp.ini. As expected, simulations with less blocks to be used by GP4SI pass off quicker: there are less possible combinations and therefore the solution-space is smaller. The results of simulations made clear that:

- If a (relatively) simple estimate of the system with only a numerator and denominator is needed that can easily be used for further calculations in Matlab, only the TF 1st-order, TF 2nd-order, a gain and an input node are to be used.

- The result of an identification is a Simulink-model. If this Simulink-model simply can be used as it results, no restrictions to the blocks to be used, have to be made. If every block is used on the other hand, the resulting GP-model mostly is a pretty complicated model.

- The differentiator-block isn’t to be used anymore because the TF 1st-order can perform the same operation and has the advantage of having no numerical problems.

An advantage of GP4SI is that one can add every built-in Simulink-function to the list of blocks GP4SI may use. At this moment one can choose out of 12 relatively basic functions.

The node-settings involve the maximum initial values, MaxInitValue, and the maximum initial stepsizes, determined by the InitStepSizeFactor, of the parameters and their optimization-algorithm. Both variables are optimized during the simulation. At first, only the parameter-values were optimized, but when using plants of different order it can be seen that the parameters vary a lot. Therefore a MaxInitValue too small or too large for the used system with a small maximum initial stepsize made it impossible for GP4SI to reach the desired parameter-values (compare simulations 14 and 15 and see simulations 17/6+7 of table B.4). As mentioned before the GA-optimization was improved and extended and now it optimizes the stepsizes of every parameter as well as the parameter-value itself. Bruijnen [3, app. C] found a value of 0.1 for the InitStepSizeFactor to work appropriately. The MaxInitValue has to be set equal to the Nyquist-frequency (half the sampling frequency) of the signal to allow GP4SI to place poles and zeros in the signal’s whole frequency-range: the MaxInitValue should be chosen about how big the maximum poles and zeros may be and has to be set in radians per second or hertz depending on the input and output-data. See for further explanation and details subsection 6.1 and Bruijnen [3, app. C] as well as the readme-file of GP4SI [4].

4 Changes to GP4SI

While working with GP4SI the results of simulations showed that some changes and additional possibilities in the program were necessary. In this section the main changes and additions that were made to GP4SI during this study and significantly influenced the obtained results are summarized. These explain some of the differences that sometimes seem to be peculiar in the results listed in appendices A and B. Besides, because of this optimization of GP4SI, most of the time-based conclusions have to be interpreted with some caution. The
successive improvements in GP4SI can be found in the readme-file of GP4SI [4].

**Additional termination criterion**

An additional termination criterion was added to make a simulation stop at a certain fitness low enough to be sure the found GP-model is sufficiently good. Now the termination criterions are: manual stop, minimal fitness and maximum number of used generations. For every plant or GP-model, the minimal fitness is different, so before setting this to a specific value, some test-simulations have to be done. Therefore its main usage lies in performing simulations with varying settings, performed on the same plant, to investigate the influences of these variations.

**Time-delay**

GP4SI is used to design a Simulink-model based on in- and output data obtained from an experimental set-up. Depending on the way of measurement, time-delay is a phenomenon that is very likely to be present to some extent. Therefore a time-delay node / block is added that can be used by GP4SI. To get a linear result with a simple numerator and denominator for further use in Matlab, this node can easily be skipped.

**TF1 and TF2**

The numerator of TF1 and TF2 can have at the most the order of the denominator. What order the numerator has, is chosen randomly at the time of creation of such a block.

**Branch addition**

Branch addition creates a new random branch and combines it with the old branch using a block with two inputs, like *add* or *subtract*; if no such blocks are selected to be used by GP4SI, the percentage of use can be set to zero. Bruijnen [3] developed this operation because it appeared to be very useful for controller design. The percentage of use is set to only 5% because the models become very complicated if a lot branches are added.

**Improvement GA-optimization**

At first only the parameter-values were optimized by the GA-optimization. The improved algorithm optimizes the stepsizes used in this optimization as well (see subsections 3.4 and 6.1 and Bruijnen [3, app. C] as well as the readme-file of GP4SI [4]). This is done per parameter of the nodes used in the GP-models.

**Over-all**

The GP-models can be forced to be stable, by allowing only poles in the left half of the S-plane. This can be useful when further simulations have to be performed with the resulting GP-model. Likewise, the zeros can be forced to be in the left half plane. This is useful when a controller has to be designed for the resulting model, because positive zeros can cause unstable closed-loop behaviour. In this study the emphasis lies on system identification and therefore this option is set to false so that non-minimum phase systems can also be identified.

5 **The fitness-function**

GP4SI is a tool that automatically tries to find a system’s dynamics by examining input-output data. Models are created as Simulink-models and simulations are performed with Matlab (see section 2). Now only the fitness / quality of a GP-model has to be determined. This is done by an m-file (*fitness.m*), which compares the output data of a GP-model with the output data of the system and assigns to each model a fitness-value: the better the model, the smaller the fitness. The algorithm that performs this evaluation is called the *fitness-function*. In fact the *fitness-function* computes a measure for the error between the GP-model and the given data. The output
data of the GP-model is derived by simulating this model with the same input data as the original system, the plant to be identified.

5.1 Time-domain based fitness-function

5.1.1 The fitness-function

The original fitness-function was designed by Bruijnen [3]. It evaluated the GP-models by calculating the difference between output data of the plant $Y_p(k,i)$ and those of the GP-model $Y_m(k,i)$. The data were based on several inputs like a step-threshold and several sine-functions with different amplitude and frequency. The fitness-function was defined as:

$$f = \sqrt{\text{mean}((Y_m(k,i) - Y_p(k,i))^2) + w(i)}$$

in which $w(i)$ is the weighting factor which indicates the importance given to the output data based on a specific input $i$ like the before mentioned threshold, sine, etc. and $\text{mean}(x)$ computes the mean of the data in a vector $x$.

The major part of the simulations used to examine the influence of the settings was performed with this time-domain based fitness-function. As a plant a 2nd-order Simulink-model (see fig. 5.3) was used so it was easy to perform simulations with various input-signals (see app. A).

Figure 5.3: The 2nd-order model, which is used as plant.

5.1.2 Results

The resulting models were validated by comparing the responses of the plant and the models on input-signals that differed from the signals used during the identification.

Figure 5.4 shows that GP4SI and the time-domain based fitness-function perform well with this specific 2nd-order plant. It can be seen that the resulting GP-model is a good representation of the plant for the given inputs, but if other inputs are used the error becomes larger very quickly. Of course GP4SI did find models capable of dealing with these inputs as well, but it can be seen easily that in this specific case the fitness-function will never be able to make a better GP-model: all the available information, the input-output data, is used as good as possible and the model does have the 'same' responses to the given inputs as the plant has. Obviously the input isn't sufficiently exciting and a more noise-like input-signal has to be used. The fitness-function has to evaluate the characteristics of the GP-models, which can only be found when a sufficiently exciting input-signal is used. The best way to show the characteristics of a system or model is by looking at their frequency response and therefore a frequency-domain based fitness-function is designed. But first a 4th-order plant was used to test the time-domain based fitness-function once more. The simulations showed that when using 4th-order plants, the fitness-function isn't capable of fitting a good model: the best results were 2nd-order approaches.

5.2 Frequency-domain based fitness-function

One of the reasons the time-domain based fitness-function didn't work appropriately, is that it doesn't try to determine the characteristics of the system but just looks at the responses on some inputs (which have to be sufficiently exciting). So models that
Figure 5.4: Validation of the result of a time-domain based fitness-function and the 2nd-order model of fig. 5.3 (simulation 6/0.2/2 of table A.2). The inputs of the left figure are used during the identification, the inputs of the right figure are used for validation.

differ a lot from the real system but with relatively low fitness based on some given, and therefore limited, input data can be found. Therefore a fitness-function that uses the frequency response function (FRF-data) of the system and GP-model, which includes all the characteristics of a system, is developed. Frequency components with low amplitude will be visible in the frequency-domain as opposed to the time-domain. The disadvantage on the other hand is that non-linear systems will be approached by linear models (non-linear behaviour is projected onto linear behaviour), while the time-domain based fitness-function can deal with both linear and non-linear systems.

The following 3 demands for the fitness-function are set:

1. The fitness-value has to be a measure that indicates to what extent two complex vectors are the same;
2. The amplitude- as well as the phase- errors have to be taken into account;
3. The fitness-value per frequency has to be a positive real number to assure that these values can be summed without cancellation of terms and that the result of the comparison of 2 vectors of complex numbers is one positive real number.

The FRF-data of a model or system can be found by calculating the power spectra of the input and output data. This can be done automatically by Matlab with the built-in tool tfe. The used input has to contain enough frequency information to produce data with good coherence and clear frequency responses. Therefore a white noise like signal with mean 0 and variable amplitude is used as input-signal. This signal is generated every fitness-determination again so that if the first generated input-signal is relatively bad, this doesn’t affect the complete identification.

To avoid cancellation of positive errors by negative ones, the errors are squared. This is done per frequency and the results are afterwards summed or averaged which leads to a single fitness-value for each GP-model. The problem arises that the errors in the amplitude and phase aren’t of the same order so they can’t be compared equally. Therefore several ways to calculate the fitness are tested.
5.2.1 Comparison of complex numbers

When looking at the pairs of complex numbers per frequency, a logical choice is to consider the vectors resulting from the differences between the numbers in these pairs. These vectors are also complex numbers and

\[
\mathbf{R} = \mathbf{C}_2 - \mathbf{C}_1
\]

These vectors are complex numbers and their length is a real number and a measure for the difference between the vectors \(C_1\) and \(C_2\).

**Figure 5.5:** left: Two complex numbers and the resulting difference vector \(\mathbf{R}\) in the complex plane. The length of \(\mathbf{R}\), \(R\), is a real number and a measure for the difference between the vectors \(C_1\) and \(C_2\). right: Maximum contribution to \(\mathbf{R}\) resulting from the phase-difference between \(C_1\) and \(C_2\) is \(R_p\), which equals 2 times the length of \(C_1\), \(2L_1\).

The length of them is dependent on the difference in length and angle between the two given vectors: the amplitude and phase errors respectively. Furthermore the length of this vector is a positive, real number and therefore it may be a good measure to compare the two complex numbers. The result of some calculation is the cosine rule:

\[
R^2 = L_1^2 + L_2^2 - 2L_1L_2 \cos(\phi_1 + \phi_2) \tag{5.2}
\]

with \(L_1\) and \(L_2\) the lengths of \(C_1\) and \(C_2\) respectively and \(\phi_1\) and \(\phi_2\) the corresponding angles of these vectors. But the result is not as good as it looks: the length of \(R\) is not limited by the amplitude-error and increases linearly with respect to the factor \(L_2\) as long as this factor is greater than 1, but the contribution of the phase-error to the length of \(R\) is at most twice the length of the smallest vector, see figure 5.5 (the right picture): the phase-error is at most 180° in which case the contribution to \(R\) equals \(2L_1\). If \(L_1\) is significantly smaller than \(L_2\), the amplitude-error will have much greater influence on the fitness-value than the phase-error, although the phase-error is maximal. This problem is enlarged by the fact that a clear distinction between the contribution of the amplitude-error respectively the phase-error to the total length of \(R\) can't be made. This is shown in equation 5.2 in which the last expression of the right argument can't be divided into a contribution of the phase-error respectively the amplitude-error. Therefore these can't be scaled in such a way that they're of the same order. Nevertheless some simulations were done to test a fitness-function on this basis. The results of these simulations with relatively simple plants proved the fitness-function to work appropriate without scaling (see table B.4, simulations 17/1-4), but it didn't work with more complex models. So to compensate the scaling problem a way to make a clear distinction between the amplitude and phase-error by scaling these errors with some sort of nonlinear boundary-function was thought of. It should have the possibility to compensate for the fact that a clear distinction between the errors can't be made and to scale the errors independently. However, in an early stage already it became clear that the introduction of such a function caused a lot of extra computation and that the phase and amplitude couldn't be compared the same over the whole range.

5.2.2 Amplitude and phase scaling

An other way is to calculate the phase and amplitude of the plant and a GP-model and compare the results. More calculation than comparing 2 columns of complex numbers is involved, but simulations show that this isn't the main part of time-consumption in \texttt{fitness.m}. Again the main problem is to scale the amplitude or phase-error in such a way that these are of the same order.
First, it has to be said that by this stage the amplitudes are already converted to a decibel or logarithmic scale: when comparing the amplitudes on a linear scale mainly the peaks in the response are emphasized while by comparing on a decibel or logarithmic scale the complete amplitude-range is taken into account. On top of that the amplitude-error is of the same order all over the used frequency-range. So for the determination of the fitness the following converted amplitudes for the model and the plant respectively are used in the fitness-function:

\[ H_{m_{dB}} = 20 \log_{10}[|H_m|] \]
\[ H_{p_{dB}} = 20 \log_{10}[|H_p|] \] (5.3)

Secondly instead of the real phase, the modulus of the phase is used: through the presence of noise, a phase tendency with a phase close to 180° or -180° may flip from the first to the latter and vice versa. This may cause very big errors if looked at the difference between the phase of the plant and a GP-model if this isn’t taken into account. Therefore the phase has to be standardized with 360°: a phase of +175° of the plant compared with a phase of -170° of the GP-model has to result in an error of 15° because the phase of the plant is very likely to be -185° instead of +175° if its tendency is around -180°. For this purpose the built-in Matlab tool mod is used and the phase-error becomes:

\[ E = \text{mod}(H_{p_{ph}}-H_{m_{ph}}+180,360)-180 \] (5.5)

with \( H_{p_{ph}} \) and \( H_{m_{ph}} \) the phase of the plant respectively the GP-model and \( E \) the absolute phase-error in degrees.

The scaling error remains. Taking the amplitude and phase of the plant and dividing the corresponding errors by these variables doesn’t work: first the amplitude as well as the phase of the plant may never become zero. This looks like a feasible assumption because it is experimental data, but the second problem is that using this scaling the size of the scaled errors depends strongly on the size of the amplitude or phase of the plant at different frequencies. If one of these is about zero, the corresponding scaled error will become very large and therefore isn’t comparable to the other error anymore. By evaluating the results of a number of simulations a scaling for the amplitude- and phase-error is experimentally designed which makes them of the same order all over the used frequency-range if they have to be of the same order (figure 5.6). Till so far the phases are measured in degrees and the amplitudes in decibels. Therefore in general the phase-errors will be larger than the amplitude-errors and the first ones are scaled by a factor \( s \). The scaling is done by looking at the errors at specific points in the responses where they are assumed to be equally bad. See for example figure 5.6: from 30 till 50Hz, around the (anti)resonances, the phase- and amplitude-errors are scaled to be of the same order at their maxima. The results of some experiments showed \( s = \frac{1}{10} \) to be a good value for the scaling-factor. This factor is also used in figure 5.6. Of course this varies per system and more or less impor-
tance to the phase- or amplitude-error can be given by setting $s$.

### 5.2.3 Coherence weighting

To prevent GP4SI from seeing noise (peaks) as system characteristics and therefore prevent GP4SI from trying to model this, a weighting with the coherence of the plant-data, $C_p$, is added to the fitness-function $F$. These data can either be experimental data or data obtained by simulating a Simulink-model. This weighting is bipartite: firstly the fitness-values of good measurement data are given more importance than the fitness-values of bad measurement data. And secondly only the samples with a sufficiently high coherence are taken into account so that really bad input-output behaviour isn’t taken into account and only the best parts of the frequency responses are fitted, the other parts are interpolated. This minimal coherence can be set by adjusting a reference value $d$, which obviously can be skipped by setting it to zero. Every sample of the input-output data of a GP-model is given a fitness-value depending on the resemblance with the corresponding input-output data of the real system. If the coherence of this corresponding data is smaller than the reference value $d$, the fitness-value of the sample becomes 0:

$$C = (C_p > d) * C_p \quad (5.6)$$

Those zero-values aren’t taken into account in the calculation of the resulting overall fitness of the GP-model. This is done by the built-in Matlab tool `find`, which filters a vector or matrix so that the zeros are excluded:

$$f = \text{find}(F * C) \quad (5.7)$$

with $F$ the original fitness-function and $C$ the coherence weighting as defined in equation 5.6. So only if the coherence of the corresponding data is larger than the reference value $d$, the error between the input-output data of the real system and the GP-model is weighted by this coherence: a high coherence means a high confidence in this error and a low coherence means a low confidence. In the calculation of the resulting overall fitness of the GP-model, the fitness-values of all samples are summarized, averaged or some other calculation the like is done.

### 5.3 Final fitness-function

The resulting fitness-function with the scaled amplitude- and phase-error and the coherence-weighting is:

$$F = ((H_{PdB} - H_{mdB})^2 + |E| * s) * C$$

$$f = \frac{\sum_{i=1}^{\text{length}(F)} \text{find}(F)}{\text{length}(F)} \quad (5.8)$$

with $H_{PdB}$ and $H_{mdB}$ the amplitudes of the plant and the GP-model respectively in decibel, $E$ as in formula 5.5 and $C$ as defined in formula 5.6. The scaling factor $s$ is taken $\frac{1}{5}$ and can be optimized at free will.

The complete file `fitness.m` can be found on the CD-rom [8]. In subsection C.2 the various ways to use this file and the involved settings are discussed.

### 6 Results

### 6.1 The need for stepsized-optimization

GP4SI and most of the used fitness-functions appeared to have no problems with the 2nd-order Simulink-model (figure 5.3), used as reference system/plant. The next step in the simulation-cycle was the identification of a 4th-order model but GP4SI had a lot of difficulties finding a good GP-model for it. The input-output data of an experimental set-up from the dynamics and control laboratory, a comparable Simulink-model and some other Simulink-models were used as plants. GP4SI appeared to have much difficulties with the (anti)resonances of the system what often resulted in a 2nd-order estimation of the 4th-order system.
By that time the GA-optimization didn’t involve the before mentioned stepsizes-optimization (see section 3.3). Therefore the stepsizes and starting values of the parameters were set at the set-up of a simulation and were the same for every parameter. The needed parameter-values for the 4th-order models were of a much higher order than those for the 2nd-order models and because of this the used MaxInitValue and MaxInitStepSize (defined in gp.ini [4]), which define the maximum start-values and stepsizes for the GA-optimization to use when optimizing the parameter-values, had to be increased a lot for some parameters of the model. Otherwise it would take too long for GP4SI to reach the needed parameter-values or they wouldn’t be reached at all.

As mentioned in paragraph 3.3 and section 4 this problem was resolved by upgrading the GA-optimization in a way it also optimizes the stepsizes during the simulation. Now there is a stepsize for each parameter of an individual and the maximum initial stepsize used in the GA-optimization is determined by the current value of that parameter multiplied by the InitStepSizeFactor defined in gp.ini. The algorithm adjusts to different orders of magnitude between parameters and can also optimize that.

Until it was clear this lack of optimization was the problem, several ideas had passed by. One of them was the use of a band-pass filter, which made the fitness-function take into account only the frequency-range where the (anti)resonances appear or give the error in this frequency-band much more importance. The major disadvantage is that before starting a simulation the data had to be plotted and some upper and lower frequency-bands had to be chosen. This meant that the black-box idea was gone and that GP4SI was going to look like some existing methods for system identification like frfit [6]. Nevertheless some simulations were performed with a band-pass filter in which only the errors in this frequency-band were included in the resulting fitness-value. But the results showed that GP4SI wasn’t able to take the (anti)resonances into account to a descent degree and moreover, the part of the response that wasn’t included in the frequency-band was even worse.

6.2 Plants of 4th- and higher order

The final fitness-function as defined in section 5.3 was tested with several plants. First a 4th-order Simulink-model was used as a plant (simulations 19/frq15 of table B.5). The results were good as can be seen in figure 6.8.

![Figure 6.7: Resulting Simulink-model of simulation 19/3 of table B.5.](image)

The resulting Simulink-model is shown in figure 6.7 and the fitness-development during the simulation in figure 6.9.

![Figure 6.9: Fitness-development of simulation 19/3 of table B.5.](image)

Next, data of an experimental set-up,
a 4th-order mass-damper-spring-mass system (figure 6.10), was used to identify this system. Again the result was good as can be seen in this figure. The resulting Simulink-model is shown in figure 6.11.

As a final test other experimental data of the same experimental set-up, now measured up to 2500 rad/s, was used. As can be seen in figure 6.12, two other eigenfrequencies of the system become visible, which means we're now dealing with an 8th-order plant. One of the difficulties of this data was the lack of an input-signal so no weighting with the coherence of the experimental data could be applied. Therefore some bandpass-filters were used to eliminate those frequency-ranges where a lot of noise was found on the output-signal or give them low confidence by a low weighting factor. From the results this absence of a good weighting seemed to be a considerable lack: GP4SI isn't able to identify the third (anti)resonance-pair and especially in this frequency-range relatively much measurement-noise is present as can be seen in figure 6.12.

The identification of the second (anti)resonance-pair appeared to be easy, but GP4SI did have much difficulties with the identification of the first and third pair. The starting values for the parameters are placed non-deterministic between zero and MaxInitValue and in this specific case the second (anti)resonance-pair covers a much larger frequency-range than the first. In fact the first pair covers a range of about 50 rad/s while the second pair covers a
range of about 800 rad/s on a total range of 2500 rad/s, which makes the chances for GP4SI to identify the first one relatively small. Therefore some simulations with a smaller MaxInitValue are performed to try to identify only the first or the second and the first (anti)resonance-pairs. This did work as can be seen in figure 6.13. The resulting Simulink-model is included in figure 6.14.

So GP4SI is capable of identifying systems on the base of input-output data using the before mentioned frequency-domain based fitness-function, but when a lot of noise is present a coherence-weighting is very useful. Furthermore, when systems of higher order are involved, the (anti)resonances may be situated in such a way that the present way of placing MaxInitValue isn’t good enough to identify all of them.

6.3 Simulation time

The main focus was to design a fitness-function that produces a model with input-output behaviour as close as possible to that of a given system. The results have to be as accurate as possible. But it would be nice if such simulation wouldn’t take days before a reasonable model is found. This wasn’t the main focus because computers become faster all the time and after all, GP isn’t a very quick way of finding solutions for problems. It is very useful because of the self-developing characteristics of the program by
which it can and does find solutions one may not have thought of. The simulations described in this paper however, took a lot of time. These simulation-times vary a lot because GP4SI makes use of a lot of non-deterministic choices during the identification, which can’t be predicted. But there are some guidelines that have to be taken into account when working with GP4SI, because the simulation time is mostly affected by:

- the number of samples used in a simulation: the more points used, the longer the simulation will take. Therefore a filter parameter for experimental data is being used, see section C.2;

- to what extent GA-optimization is used (see also 3.3) because this is the part of a fitness-evaluation that takes longest.;

- the number of different nodes used: the less nodes used, the shorter the simulation takes. This as a result of a smaller solution space, see section 3.4.

- the maximum treedepth that is chosen: the smaller MaxTreeDepth, the simpler the Simulink-models will be and the faster the simulation of these models will go.

Basically every setting influences the simulation time, but the above mentioned ones are the most dominating.

7 Conclusions and recommendations

Conclusions

Conclusions involving the settings-optimization:

- As discussed in subsection 3.4 depending on the desired result: whether a Simulink-model or a transfer-function for further use in Matlab-calculations of the system is desired. In the latter case, just a few blocks have to be used, while in the first case the complexity of the model isn’t an issue and every block may be used.

- The generally used percentages for the genetic operations as mentioned in subsection 3.3 perform well and may be taken slightly higher or lower when identifying more respectively less complicated systems.

- To gain more variation in the models and prevent GP4SI from coming to a deadlock because no models with a better fitness are found anymore, the MaxTreeDepth and the number of Individuals may be enlarged.

The settings that are generally used in this study are included in gp.ini [8].

Conclusions involving the redesign of the fitness-function:

- The time-based fitness-function performs well when relatively simple systems are involved. For more difficult systems a frequency-based fitness-function performs better.

- The fitness-function as defined in subsection 5.3 works for higher order systems. But because of the frequency-based fitness-evaluation, non-linearities will be approached by linearities and can’t be identified with this fitness-function.

- If a frequency-domain based fitness-function is used, a noise-signal is a logical choice to use as input. Because it is a noise-signal, during a simulation it can be regenerated every identification of an individual again. So the original input-signal is only used in the computation of the coherence for the
coherence-weighting. This coherence-weighting is very useful, so the original input data, of course as well as the output data, are necessary for a good identification.

Overall conclusions:

- Good settings and a well-chosen fitness-function aren't guarantees for GP4SI to work. It is a stochastic process and in some cases a good system identification is made and in other cases not (compare simulations 4/1, 2 with 4/3 of table A.1: the same settings and fitness-function, but in the latter case the final fitness isn't reached although it is in the other 2 cases). If a simulation has ran long enough, the major part of the population may have converged to a local minimum and a lack of diversity among the individuals may cause a deadlock. A larger population of individuals may help, but the examination-time of 1 generation will increase significantly.

- If so-called 2-inputs/1-output blocks like a sum or divider are used, the GP-models can become relatively complicated. The maximum number of blocks in a branch is limited by the MaxTreeDepth, but it is important that GP4SI finds a relatively good model in an early stage because the GP-models will become increasingly complicated during the identification and simulating these models, which is needed for the fitness-evaluation, will take more and more time. So if a good system identification still hasn't been made after a day or so, you may think of starting a new one.

Recommendations

Some recommendations for further research and study are:

- More simulations with the final fitness-function of subsection 5.3 using experimental set-ups of 4n or higher order and greater complexity have to be performed to test and optimize it.

- Because non-linearities can't be identified in the frequency-domain, the time-domain based fitness-function may be redesigned or better optimized than the tests of section 5.1.

- As mentioned in subsection 3.2, the influences of the use of not only tournamentselection and other settings for the SelectionAmount have to be examined.

- To enlarge the black-box idea of GP4SI, the tool may be extended with an user-interruptible automatic determination of the bandwidth of the system to be identified. By means of that the MaxInitValue, which is the upper boundary for the initial parameter-values, can be updated automatically and is always suitable for the used system.

- An other phase- and amplitude-scaling, which is automatically set for every plant may be a better solution to the scaling-problem.

- Search for other parameter-optimization methods (see Bruijnen [3]); the used GA-optimization consumes most of the simulation-time.
8 References


[4] D. Bruijnen, Genetic Programming for System Identification (GP4SI), a GP-program for system identification designed as part of a study on automated controller design using GP [3], including:
- GP4SI.exe , a tool for System Identification using GP
- gp.ini , a textfile including the settings of GP4SI
- calcsstop.dll , a stop criterion
- protecteddiv.dll , a substitute for the Simulink built-in divider-block
- readme.doc , a brief explanation of the working and settings of GP4SI
- fitness.m , the Matlab-file containing the fitness-function


[8] G. Naus, simulations performed with GP4SI, a CD-Rom accompanying this study, including also the latest version of GP4SI (see [4]), 2003.


A  Settings optimization

In the following tables the results of the simulations performed for the optimization of the settings are listed. The simulations are performed using a 2nd-order Simulink-model as plant (see section 5). Not all simulations are included because of the unfitness of some of them and besides during the optimization of the fitness-function also some variation in settings is tested but these simulations are included in the tables of appendix B. Each column contains a specific simulation-parameter. These are respectively:

1. every simulation is numbered. Besides, the category refers to the appropriate folder on the CD-rom where more information about every simulation can be found and the results of the simulations can be regenerated. In most categories various simulations are performed as to obtain a reliable result;

2. the total number of generations used by GP4SI for a simulation;

3. the end fitness is the final fitness after a simulation;

4. t-time stands for the total time the simulation took;

5. to compare the simulations a minimal fitness-value for the 2nd-order model used in these simulations has been determined. The variables in this column show the number of evaluations GP4SI used before it reached this minimal fitness-value;

6. total number of fitness-evaluations GP4SI performed in this simulation;

7. the mean number of evaluations per hour, calculated by means of the total simulation-time (column 4) and the total number of fitness-evaluations (column 6);

8. w-time stands for the time it took GP4SI to find a GP-model with a fitness-value low enough to be sure to be a good model. This variable is estimated by means of the number of used evaluations it took GP4SI to find this GP-model (column 5) and the evaluations per hour (column 7) in this simulation;

9. the used version of GP4SI is mentioned (see the CD-Rom [8]) and besides some remarks involving the appropriate simulation are noted.

In the tables A.1 and A.2 simulations performed with a time-based fitness-function are listed. In table B.3 the simulations performed with a frequency-based fitness-function are listed.
### Table A.1: Settings optimization 1 - simulations with a time-based fitness-function

<table>
<thead>
<tr>
<th>cat.</th>
<th>gen.</th>
<th>end-fitn.</th>
<th>t-time</th>
<th>fitn&lt;0.23</th>
<th>fitn.eval.</th>
<th>eval./h</th>
<th>w-time</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>optimisation procedures 1; GA-opt: 1.0 and Nelder-Mead-opt: 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GP4SI 1.1</td>
</tr>
<tr>
<td>opt1</td>
<td>318</td>
<td>0.221</td>
<td>51.28h</td>
<td>111696</td>
<td>161102</td>
<td>3142</td>
<td>35.6h</td>
<td>converges very slowly</td>
</tr>
<tr>
<td>2</td>
<td>optimisation procedures 2; GA-opt: 0.5 and Nelder-Mead-opt: 0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GP4SI 1.1</td>
</tr>
<tr>
<td>opt2</td>
<td>177</td>
<td>0.220</td>
<td>24.8h</td>
<td>25956</td>
<td>45185</td>
<td>1822</td>
<td>14.2h</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>417</td>
<td>0.260</td>
<td>24.3h</td>
<td>39737</td>
<td>43655</td>
<td>1797</td>
<td>22.1h</td>
<td></td>
</tr>
</tbody>
</table>

| 3    | nodes 1; only TF2, gain and input are used | | | | | | | GP4SI 1.1 |
| nod1 | 851  | -         | 1.20h  | -         | 5313       | 4119    | -     |
| 2    | 147  | 0.233     | 6.10h  | 10207     | 10207      | 1672    | 6.1h  |
| 3    | 408  | 0.218     | 16.85h | 8641      | 29848      | 1771    | 4.9h  |
| 4    | 606  | 0.217     | 23.51h | 18177     | 40901      | 1740    | 10.4h |

| 4    | nodes 2; only TF2, gain, add, subtract, constant and input | | | | | | | GP4SI 1.1 |
| nod2 | 231  | 0.227     | 4.82h  | 10292     | 10825      | 2246    | 4.6h  |
| 2    | 457  | 0.217     | 23.38h | 9176      | 30799      | 1313    | 7.0h  |
| 3    | 2209 | 2.839     | 26.43h | -         | 118339     | 4478    | -     |

| 5    | Koza; crossover 95%, both mutations 5% | | | | | | | GP4SI 1.1 |
| Koza | 401  | 1.868     | 7.4h   | -         | 13514      | 1826    | -     |
| 2    | 1255 | 2.036     | 24.29h | -         | 40574      | 1671    | -     |
| 3    | 713  | 1.839     | 25.96h | -         | 41649      | 1738    | -     |

### Table A.2: Settings optimization 2 - simulations with a time-based fitness-function

<table>
<thead>
<tr>
<th>cat.</th>
<th>gen.</th>
<th>end-fitn.</th>
<th>t-time</th>
<th>fitn&lt;0.23</th>
<th>fitn.eval.</th>
<th>eval./h</th>
<th>w-time</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>optimisation procedures 3; varying the GA-optimization from 0.2 till 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GP4SI 1.3</td>
</tr>
<tr>
<td>opt 0.2</td>
<td>162</td>
<td>0.217</td>
<td>18.91h</td>
<td>11181</td>
<td>29840</td>
<td>1578</td>
<td>7.1h</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>0.230</td>
<td>16.53h</td>
<td>17656</td>
<td>17656</td>
<td>1068</td>
<td>16.5h</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>205</td>
<td>0.217</td>
<td>2.24h</td>
<td>6199</td>
<td>56096</td>
<td>25037</td>
<td>0.2h</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2586</td>
<td>0.236</td>
<td>24.87h</td>
<td>518280</td>
<td>518280</td>
<td>20854</td>
<td>24.9h</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2112</td>
<td>0.217</td>
<td>20.29h</td>
<td>420522</td>
<td>20736</td>
<td>20736</td>
<td>2.1h</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>1</td>
<td>95</td>
<td>0.411</td>
<td>25.47h</td>
<td>-</td>
<td>36521</td>
<td>1434</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>0.229</td>
<td>16.30h</td>
<td>37640</td>
<td>37019</td>
<td>2313</td>
<td>16.3h</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>113</td>
<td>0.217</td>
<td>2.26h</td>
<td>23896</td>
<td>49031</td>
<td>21727</td>
<td>1.1h</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1210</td>
<td>0.217</td>
<td>24.88h</td>
<td>30721</td>
<td>490065</td>
<td>19097</td>
<td>1.6h</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>1</td>
<td>82</td>
<td>0.217</td>
<td>24.34h</td>
<td>52230</td>
<td>52265</td>
<td>2200</td>
<td>16.3h</td>
</tr>
<tr>
<td>2</td>
<td>89</td>
<td>0.220</td>
<td>2.27h</td>
<td>49036</td>
<td>53936</td>
<td>23370</td>
<td>2.1h</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>809</td>
<td>0.217</td>
<td>24.91h</td>
<td>36060</td>
<td>481259</td>
<td>19323</td>
<td>1.9h</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>613</td>
<td>0.217</td>
<td>20.22h</td>
<td>160692</td>
<td>386471</td>
<td>19112</td>
<td>8.4h</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>1</td>
<td>92</td>
<td>0.223</td>
<td>16.36h</td>
<td>71361</td>
<td>71931</td>
<td>4307</td>
<td>16.2h</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>0.329</td>
<td>2.23h</td>
<td>-</td>
<td>48382</td>
<td>21677</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>563</td>
<td>0.217</td>
<td>24.92h</td>
<td>98504</td>
<td>455914</td>
<td>18294</td>
<td>5.4h</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>443</td>
<td>0.216</td>
<td>20.18h</td>
<td>178404</td>
<td>397763</td>
<td>19711</td>
<td>9.1h</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>35</td>
<td>0.225</td>
<td>16.44h</td>
<td>32980</td>
<td>33891</td>
<td>2062</td>
<td>16.0h</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>1.052</td>
<td>2.24h</td>
<td>-</td>
<td>52865</td>
<td>25646</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>377</td>
<td>0.217</td>
<td>18.63h</td>
<td>26062</td>
<td>401523</td>
<td>21547</td>
<td>1.2h</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>492</td>
<td>0.216</td>
<td>24.88h</td>
<td>57247</td>
<td>510444</td>
<td>20515</td>
<td>2.8h</td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Settings optimization 2 - simulations with a time-based fitness-function
B Fitness-function optimization

In the following tables the results of the simulations performed for the optimization of the frequency-based fitness-function are listed. Each column contains a specific simulation-parameter. The first columns are the same as in appendix A, the others are:

- the data or Simulink-models that are used as plant; `plant_4th_order` stands for `plant_4th_order` and `data_1` for the data derived experimentally from a 4th-order system. These plants are included on the CD-Rom [8];
- total number of used data points;
- the minimal coherence of the plant allowed during the simulation;
- the results of the used fitness-function and settings. If nothing is given, the result of the simulation was bad.

The columns of the first table differ slightly from the other tables. The used (frequency-based) fitness-functions corresponding to the simulations are listed as title for every new block of simulations. In the table B.4 the simulations performed with the different frequency-based fitness-functions are listed.

<table>
<thead>
<tr>
<th>cat.</th>
<th>gen.</th>
<th>end-fitn.</th>
<th>t-time</th>
<th>fitn&lt;0.02</th>
<th>fitn.eval.</th>
<th>eval./h</th>
<th>coh</th>
<th>results</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GP4SI 1.2</td>
</tr>
<tr>
<td>freq1</td>
<td>1</td>
<td>518</td>
<td>23.1h</td>
<td>7711</td>
<td>30716</td>
<td>1330</td>
<td></td>
<td>reasonably good model</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>652</td>
<td>28.1h</td>
<td>40709</td>
<td>40709</td>
<td>1449</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1398</td>
<td>43.88h</td>
<td>49149</td>
<td>84633</td>
<td>1929</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

| 8    | F    |           |        |           |            |        |     | GP4SI 1.2 |
| freq2| 1    | 1294      | 25.67h | 63614     | 2478       | 0.8    |     | |
|      | 2    | 1268      | 25.01h | 63123     | 2465       | 0.8    |     | |

| 9    | F    |           |        |           |            |        |     | GP4SI 1.2 |
| freq3| 1    | 991       | 21.17h | 54651     | 2581       | 0.5    |     | |
|      | 2    | 878       | 19.38h | 50080     | 2585       | 0.5    |     | |
|      | 3    | 925       | 19.41h | 52167     | 2687       | 0.5    |     | |

| 10   | as 8, only TF2, gain, input, GA-opt: 0.5, Nelder Mead-opt: 0.5, no C_max | GP4SI 1.2 |
| freq4| 1    | 145       | 17.52h | 37196     | 40556      | 2315   |     | |
|      | 2    | 138       | 17.52h | 34703     | 37443      | 2137   |     | |

| 11   | same as 8, but with random input apart and no C_max | GP4SI 1.2 |
| freq5| 1    | 323       | 18.69h | 7852      | 19155      | 1025   |     | |
|      | 2    | 256       | 16.89h | 14876     | 881        | 0.5    |     | |

Table B.3: Fitness-function optimization - simulations using the 2th-order model
**Table B.4: Fitness-function optimization - simulations with a frequency-based fitness-function**

<table>
<thead>
<tr>
<th>cat.</th>
<th>gen.</th>
<th>end-fn.</th>
<th>t-time</th>
<th>fitn-eval.</th>
<th>eval./h</th>
<th>plant</th>
<th>pnts</th>
<th>coh</th>
<th>results</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>freq7</td>
<td>1</td>
<td>668</td>
<td>0.047</td>
<td>24.90h</td>
<td>53926</td>
<td>6141</td>
<td>pL2nd_ord</td>
<td>2000</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>711</td>
<td>0.417</td>
<td>19.24h</td>
<td>16069</td>
<td>8348</td>
<td>pL2nd_ord</td>
<td>2000</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1016</td>
<td>0.325</td>
<td>24.92h</td>
<td>230836</td>
<td>9264</td>
<td>pL2nd_ord</td>
<td>2000</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>213</td>
<td>0.354</td>
<td>5.23h</td>
<td>46867</td>
<td>8965</td>
<td>pL2nd_ord</td>
<td>2000</td>
<td>0.95</td>
</tr>
</tbody>
</table>

| 13   |      |         |        |            |         |       |      |     |         |
| freq8| 1    | 588     | 0.079  | 16.75h    | 90965   | 5395  | pL2nd_ord | 2000 | 0.95 | good model |
|      | 2    | 597     | 0.084  | 16.69h    | 129337  | 7752  | pL2nd_ord | 2000 | 0.95 | good model |
|      | 3    | 322     | 0.118  | 16.64h    | 71991   | 4326  | pL2nd_ord | 2000 | 0.95 | good model |
|      | 4    | 420     | 0.099  | 16.84h    | 97614   | 5795  | pL2nd_ord | 2000 | 0.95 | good model |

| 14   |      |         |        |            |         |       |      |     |         |
| freq9| 1    | 473     | 71.071 | 21.11h    | 110953  | 5256  | pL2nd_ord | 2000 | 0.95 | reasonably mdl |
|      | 2    | 134     | 119.761| 4.54h     | 26553   | 5843  | pL2nd_ord | 2000 | 0.95 | good model |
|      | 3    | 124     | 34.945 | 4.55h     | 25845   | 5725  | pL2nd_ord | 2000 | 0.95 | good model |
|      | 4    | 384     | 24.080 | 16.61h    | 91740   | 5522  | pL2nd_ord | 2000 | 0.95 | good model |

| 15   |      |         |        |            |         |       |      |     |         |
| freq10| 1   | 983     | 29.662 | 71.77h    | 236596  | 3296  | pL4th_ord | 2000 | 0.95 | good model |
|      | 2   | 1106    | 17.017 | 71.75h    | 241732  | 3589  | pL4th_ord | 2000 | 0.95 | good model |
|      | 3   | 974     | 43.858 | 71.75h    | 229819  | 3189  | pL4th_ord | 2000 | 0.95 | good model |
|      | 4   | 832     | 681.808| 71.75h    | 203737  | 2839  | pL4th_ord | 2000 | 0.95 | good model |

The used 4th-order model was the same as used by D. Bruijnen [4], but it appeared to be to difficult to distinguish at this point.

| 16   |      |         |        |            |         |       |      |     |         |
| freq11| 1   | 520     | 0.401  | 17.50h    | 115952  | 6835  | pL4th_ord | 4000 | 0.95 | good model |
|      | 2   | 635     | 0.388  | 17.33h    | 130331  | 8042  | pL4th_ord | 4000 | 0.95 | good model |

| 17   |      |         |        |            |         |       |      |     |         |
| freq12| 1   | 96      | 6.190  | 2.23h     | 16293   | 7307  | pL2nd_ord | 4000 | 0.90 | good model |
|      | 2   | 644     | 32.161 | 24.13h    | 144749  | 5999  | pL2nd_ord | 4000 | 0.90 | good model |
|      | 3   | 583     | 4.298  | 18.51h    | 142188  | 7883  | pL2nd_ord | 4000 | 0.90 | good model |
|      | 4   | 596     | 23.445 | 18.58h    | 121822  | 6558  | pL2nd_ord | 4000 | 0.90 | good model |

The used 4th-order model was the same as used by D. Bruijnen [4] but it appeared to be too difficult to distinguish at this point.

| 18   |      |         |        |            |         |       |      |     |         |
| freq13| 1   | 400     | 299.464| 9.025h    | 294392  | 3262  | data_1 (4th) | 15000 | 0.20 | pretty good mdl |
|      | 2   | 286     | 247.796| 11.10h    | 207759  | 1744  | data_1 (4th) | 30000 | 0.20 | pretty good mdl |
|      | 3   | 332     | 570.330| 11.09h    | 264704  | 2223  | data_1 (4th) | 20000 | 0.20 | pretty good mdl |

The scaling wasn't correct because the phase was taken in radians instead of degrees.
Table B.5: Fitness-function optimization - simulations with a frequency-based fitness-function
C  Users manual

In this manual the steps to make and the files involved when using GP4SI for system identification of a system are discussed. Guidelines for the settings of gp.ini are summarized in the conclusion, the standard settings can be found on the CD-Rom [8] and the possibilities of the present fitness-function are discussed in subsection C.2. An extensive explanation and description of the used C++ code, every file and some troubleshooting can be found in Brujin [3, app.C] where he also included a users manual for GP4CD. A lot of the items in this appendix are taken from that manual but adapted to system identification with GP4SI.

C.1 Involved files

First the standard files that come with GP4SI are discussed and next the files that are created as a result of a simulation with GP4SI. The files can be found on the GP4SI CD-rom [8].

C.1.1 Standard GP4SI files

First of all a readme.doc file is present with a brief explanation of the use of GP4SI and a description of the settings to be set in gp.ini. Furthermore the standard settings are presented.

  calcstop.dll
A S-function that is used in every Simulink GP-model and terminates the simulation if a maximum running time is reached. It prevents the simulation from trying on forever to calculate the fitness of very complex models that may be (close to) singular. Such models have to be ignored and are given a bad fitness.

  protecteddiv.dll
To prevent simulations from interrupting upon a division by zero. This file is used instead of the division blocks in the Simulink GP-models. The denominator saturates if the absolute value is smaller than 1e-6 so it will not become 0.

  gp.ini
The settings used in GP4SI are determined and can be set in this file. More about these settings can be found in section 3 and appendix C. The standard settings can be found in readme.doc [8].

  fitness.m
The Simulink-models made by GP4SI are evaluated by this Matlab-function. The fitness of the models is calculated with the fitness-function (see C.2).

  GP4SI.exe
The executable file that starts the GP-run. Before execution, the settings in gp.ini have to be considered and the fitness-function must be set up correctly. After that the simulation can be started by running GP4SI.exe and you just have to wait for a good solution that minimizes the fitness-function.

C.1.2 Resulting files after a simulation

During a simulation GP4SI creates GP-models that are tested on fitness. Every time a new best model is found, it will be saved in a 'mdls'-subdirectory created by GP4SI. At the end the final GP-model and its settings will be saved with a user-defined name in a user-defined
subdirectory. The resulting files are listed in this subsection.

\texttt{temp\_modelxxx.mdl}

Every Simulink GP-model with the best fitness-value so far will be saved in the 'mdls'-subdirectory as \texttt{temp\_modelxxx.mdl} with xxx the successive numbers. This are all temporary models, created during the simulation; the final GP-model will be saved with a user-defined name.

\texttt{fitness.log}

In this file the fitness and fitness evaluation count of each new best GP-model is saved. With this the fitness progression and the accompanying models when (substantial) fitness improvements occur can be examined afterwards.

\begin{itemize}
  \item \texttt{name\_loghistory.txt}
  \item \texttt{name\_fitness.log}
  \item \texttt{name\_.mdl}
  \item \texttt{name\_fitness.m}
  \item \texttt{name\_plant.mdl}
\end{itemize}

The files listed above will be saved at the end of a simulation with \texttt{name} a user-defined name. The best GP-model, the \texttt{fitness.log} file, the used fitness-function and the used system if available (see sect. ?? are copied to a user-defined subdirectory. Furthermore the settings are summarized in \texttt{loghistory.txt} so in a later stage a similar simulation can be performed.

C.2 Guidelines for setting the fitness-function

Based on the available input-output signals and the sort of plant to be identified generally 3 ways to use \texttt{fitness.m} are possible. If one of the available time- or frequency-based fitness-functions is used, only a few settings have to be set.

1. Only some experimental input-output data is available. Dependent on the used input-signal the time-based or frequency-based fitness-function can be used. In most cases off course this will be a noise-a-like signal to construct an amplitude- and phase-respon- of the system and the frequency-based fitness-function has to be used. The most important settings are the minimal coherence, \texttt{coh}, that is allowed, the number of blocks, \texttt{fp1}, used when calculating the FRF-data using \texttt{tfe}, and the number of points used of the whole data-range. Often it isn’t smart to use all data-points in the simulation because the GP-models have to be simulated with as many points and this costs a lot of time. Therefore the filterparameter \texttt{fp2} has to be set: one out of \texttt{fp2} points is used.

2. A Simulink-model is used as a plant and has to be identified (as a test). In this case both the time-based and the frequency-based fitness-function can be used because every input-signal can be tested and the responses of the plant can be measured easily. When using the time-domain fitness-function the most important settings are the input-signals and the number of used points. Using the frequency-based fitness-function, the same points mentioned in 1 are of importance.

3. An experimental set-up has to be identified and it is relatively simple to measure and test some input-output signals and behaviour. Dependent on the complexity of the system the time- or frequency-based fitness-function can be chosen.
The resulting `fitness.m` can be found on the CD-Rom [8] and before starting a simulation, the choice whether the time-domain or the frequency-domain fitness-function is used has to be made and the corresponding programlines have to be uncommented.

### C.3 Performing a simulation with GP4SI

1. Adjust the fitness-function for the type of simulation you want to run and adjust the settings involved.

2. Check all settings of `gp.ini` and adjust them where necessary.

3. Run `GP4SI.exe` and wait. It may take some time, sometimes rather a day than some hours, but it will improve equally with your workstation. During the simulation you can evaluate the temporary results.

4. When a termination criterion is reached (for example manual interruption), the results can be saved with a user-defined name in a user-defined subdirectory.

5. Finally the saved results can be evaluated and validated.