Hongesombut K., Mitani Y., Tsuji K.:

An Incorporated Use of Genetic Algorithm and a Modelica Library for Simultaneous Tuning of Power System Stabilizers

2nd International Modelica Conference, Proceedings, pp. 89-98

An Incorporated Use of Genetic Algorithm and a Modelica Library for Simultaneous Tuning of Power System Stabilizers

Komsan Hongesombut, Yasunori Mitani, and Kiichiro Tsuji

Osaka University, Graduate school of engineering
2-1 Yamada-oka, Suita, Osaka 565-0871, JAPAN

Abstract

ObtectStab package that has been successfully applied to power system studies is a general-purpose simulation tool developed by the Modelica language. It takes advantages from the capability of physical modeling of Modelica language that make ones readily develop new models and use them for complex and large cases of power system studies based on object-oriented programming. However, in the situation that control of complex power systems is not easy to be realized by traditional methods, genetic algorithm (GA) becomes an alternative powerful method that can be used to solve several difficult problems without any prior or little knowledge of the systems being solved. Proposed in this paper is an incorporating the use of GA to an ObjectStab library to enhance the use of this library into optimization environment. The idea has been applied to one challenging problem of simultaneous tuning power system stabilizers in a multimachine power system. The simulation results show that the resulting controller obtained by a GA can achieve good performance.

Index Terms – ObjectStab, genetic algorithms, Simulink interface, simultaneous tuning, power system stabilization.

1. Introduction

Until recently, there has been widespread interest using genetic algorithms (GA's) to search and optimize in several difficult problems. Compared to traditional search and optimization procedures, such as calculus-based approach, GA's are robust, conceptually simple to apply in problems where little or no prior knowledge is available for the problem being solved. Problems on modern power systems are more and more difficult to be solved by using only conventional techniques due to large complex networks and nonlinear characteristic of power systems. The need of using other alternative tools such as genetic algorithms to solve such difficult problems become evident in case many conventional techniques get into difficulties. Incorporating the use of GA and power system simulation tools, among them such as PSCAD/EMTDC, EMTP, EuroStag, etc, ObjectStab [1] in Dymola [2] which is a library developed by Modelica language [3] for power system studies is more flexible than those in the view point of its easiness to realize the phisical models and its powerful interface with MATLAB and Simulink that can allow ObjectStab be used with optimization methods such as GA. This paper describes a method of how a GA can be applied to a Modelica library named ObjectStab. An example of simultaneous tuning of power system stabilizers in a multimachine power system is used to validate the effectiveness of the incorporated use of these two features. It opens up a new idea of the use GA and Modelica library together allowing designers to design more sophisticated controllers. The idea does not limit only the applications to power systems, but also other Modelica users can adapt this idea to their own works. The simulation tools used in this paper are the Dymola, ObjectStab library, MATLAB [4] and Simulink [5] and Genetic and Evolutionary Algorithm Toolbox (GEATbx) [6].

2.Genetic Algorithms

A Genetic algorithm (GA) is a biologically inspired search algorithm pioneered by Holland [7]. The approach is based on Darwin's survival of the fitness hypothesis. In GA's, candidate solutions to a problem are analogous to individuals in a population. A population of individuals is maintained within search space for a GA, each representing a possible solution to a given problem. The initial population can be a random collection of bizarre individuals. The individuals will interact and breed to form future generations (offspring). The stronger individuals will reproduce more often than weaker individuals. Presumably, the population will get collectively stronger as generations pass and weaker individuals die out. Unlike other optimization methods, GA's do not limit by constraints on the form of fitness function. The fitness function does not need to be differentiable or continuous. This flexibility in which GA's use a fitness function to search for the solution makes GA's become a power tool for optimization in many difficult problems in many fields.

GA's work with coding of the parameters themselves (called string) and then use the genetic operators to evolve the solution with minimum computation. An optimal solution can be found and represented by the final winner in the competitive environment. GA's consist of simple three operators; selection, crossover and mutation. Selection is the operation in which the fittest individual of the population in the current generation forms part of the population to the new generation. Crossover is responsible for providing new offspring by selecting two individuals and exchanging some parts of their structures. Mutation is an operator which is applied for altering the value of a random position in a string. A simple algorithm flowchart is shown in Fig.1.

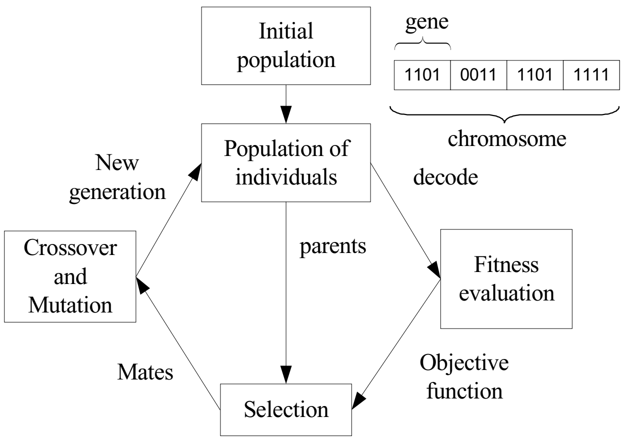


Fig.1 Simple algorithm flowchart of GA

3.Combination of Modelica library and Genetic Algorithms

In this section, we will generally describe how a Modelica library combines with a GA. One of the most powerful features of MEX files, including C format S-functions is it allows ones to incorporate existing code into a Simulink model. The key idea of combination a Modelica library and GA is using this feature by converting a Modelica model to a compiled MEX-file used in Simulink as an S-function block. Then a GA that exists in MATLAB environment will adjust some parameters of a Modelica model according to the fitness values. Briefly, incorporating a Modelica library and GA can be achieved by these following steps:

1. Build a Modelica model. The model is build up in Dymola environment.
2. Build a Simulink model named model 1 by using a DymolaBlock which is a new interface to Simulink that can be found in Simulink's library browser. This block is shielded around an S-function MEX block that interfaces to the C code generated by Dymola for the Modelica model. Model 1 is constructed for serving as an interfacing block for editing and compiling for two environments by switching the current active window between Dymola and Simulink environment.
3. Compile to Simulink dll file. It is possible to converted a Modelica model to a compiled MEX-file *SimStr.dll* to be used as one block in Simulink environment. By doing this, command *dymcomp* is used.
4. Build a Simulink model named model 2. This model is served as a main system for connecting with a GA. It contains an S-function block representing a model as in Dymola and Simulink model for calculating fitness values used in a GA. Parameters and initial conditions are be defined or changed by passing these variables as inputs to S-function block.
5. Build a main m-file and a function used in a GA.

Details of above procedures are summarized and given in Fig.2. After this short summary of how a Modelica model combines with a GA, we will continue by real building a model for simultaneous tuning PSSs in a multimachine power system. We will show the flexibility of using GA by using two objective functions with the same Modelica model.

4. Problem Formulation

The objective of this problem is to tune an appropriate set of PSSs to damp local and inter-area modes. This problem is not easy by using traditional analytical methods to simultaneously tune all PSSs. The fixed structure of i^{th} PSS as shown in equation 1 is used for all 4 generators. It consists of a two-stage lead lag compensation with time constants T_{1i} - T_{4i}, and a gain K_i. We set the wash out time constant T_{wi} with large enough value so that it can be considered as a constant.

$$PSS(s)_i = K_i \frac{sT_{wi}}{1+sT_{wi}} \left(\frac{1+sT_{1i}}{1+sT_{2i}} \frac{1+sT_{3i}}{1+sT_{4i}} \right) \qquad (1)$$

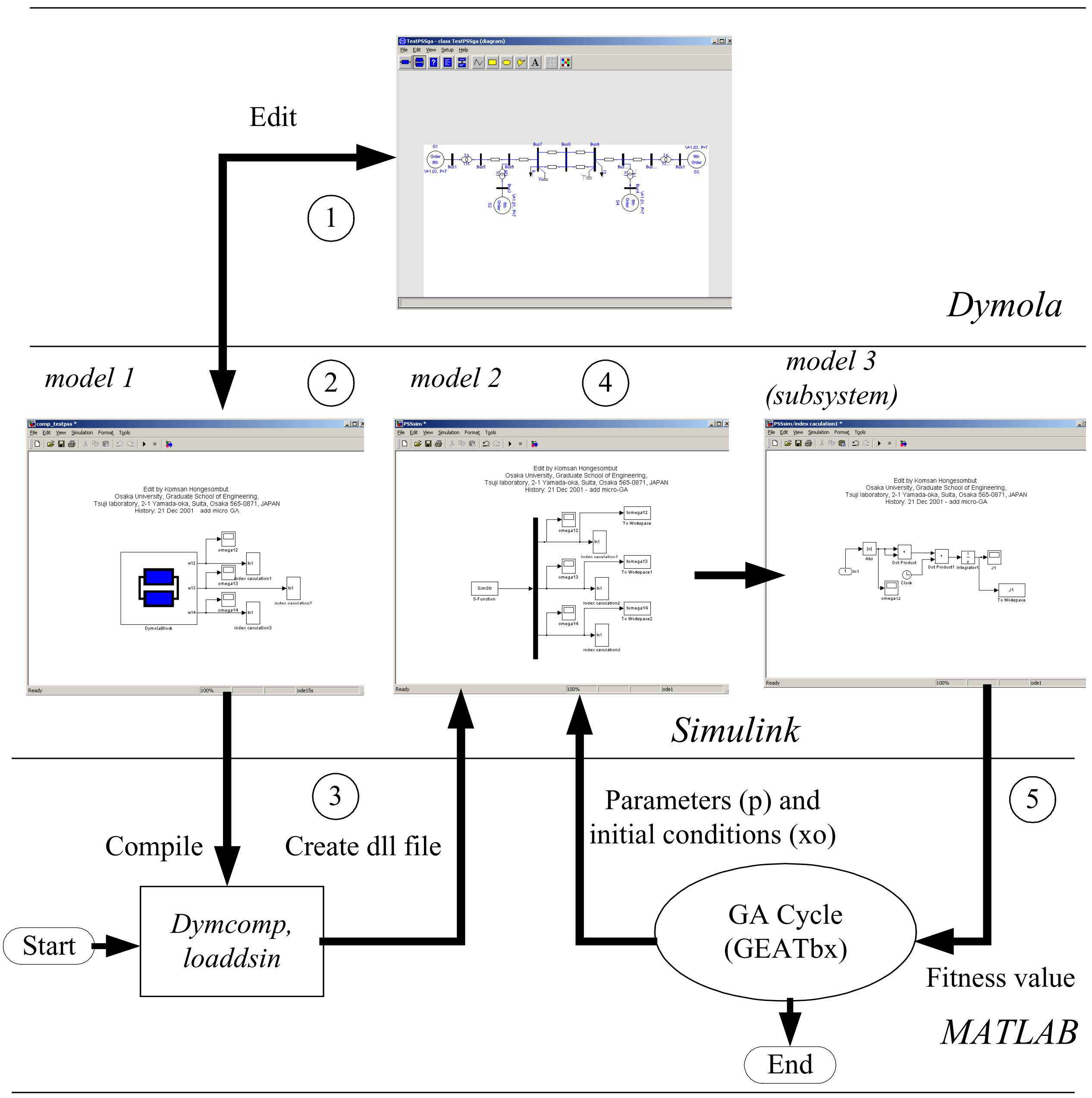


Fig. 2 Summary of how to combine a Modelica library with a GA

where

$$T_{1i} = T_{3i} = \frac{\sqrt{\gamma_i}}{\delta_i} \quad \text{and} \quad T_{2i} = T_{4i} = \frac{1}{\delta_i \sqrt{\gamma_i}} \qquad (2)$$

We present two methods to satisfy the objective of tuning PSS as follows,

4.1 Method 1: time domain-based performance index

Typically, the performance of the design controller is measured directly from the output responses varying with time. This is a straightforward approach that can guarantee the performance of controllers under scenarios which are predefined by the designer. Equation 3 shows the objective function used in a GA meaning that we are trying to minimize the deviation of generator speed for local and inter-area modes by applying the suitable set of PSS control parameters.

$$\min F = \int_{t=0}^{10} \left(\left|\Delta\omega_{12}\right|^2 + \left|\Delta\omega_{13}\right|^2 + \ldots \left|\Delta\omega_{1n}\right|^2 \right) \cdot t\, dt \qquad (3)$$

where n is the number of generators by assuming that generator 1 is a reference.

4.2 Method 2: eigenvalue-based performance index

For every operating conditions under consideration, here, it is supposed that a linearized model of power system is obtained first. The problem of selecting the parameters for power system stabilizers that can assure minimum damping performance over the considered set of operating point is converted to a simple optimization problem and then is solved by a GA with an eigenvalue-based performance index. The GA objective function is derived in this following way:

A linear model of power system is extracted around a certain operating condition. The system can be expressed in the linear state-space form as shown in the following equations

$$\dot{x} = Ax + Bu \qquad (4)$$

$$y = Cx + Du \qquad (5)$$

The equation expressed for the controllers is shown in (6) where in this study, the controller *K(s)* is a lead-lag type that is the same as described by the transfer function in (1). *y(s)* is the measuring signal and *V(s)* is the output signal from the controller which provides additional damping by shifting under damped or unstable oscillation modes to the left hand side of the s-plane.

$$V(s) = K(s)y(s) \qquad (6)$$

Combining equation 4 through 6, a closed-loop eigenvalues of the system can be obtained. Here, let $\lambda_i = \alpha_i \pm j\beta_i$ be the i^{th} mode of the closed-loop system. Damping coefficient δ_i of the i^{th} mode is calculated by

$$\delta_i = -\frac{\alpha_i}{\sqrt{\alpha_i^2 + \beta_i^2}} \qquad (7)$$

If *p* is a number of operating conditions where each condition contains the matrix of damping coefficient δ_i, *i = 1, …, n* where *n* is the number of oscillation modes of the closed-loop system. The optimization problem to be solved by a GA can be written in the following form:

$$\max F = \min(\min(\delta_i))_p \qquad (8)$$

For simplicity, we will choose only one operating condition for considering in this paper.

5. Test Power System and Scenarios

Fig.3 shows a single line diagram of a test power system constructed by using a graphical editor of Dymola and ObjectStab. The data of this power system network is given in [8]. The disturbance considered in this study is a three-phase to ground fault near Bus 7 by the following situations:

t = 1 s : fault is applied,

t = 1.1 s : fault is cleared by tripping one of two parallel lines.

t = 2.5 s : line is reclosed.

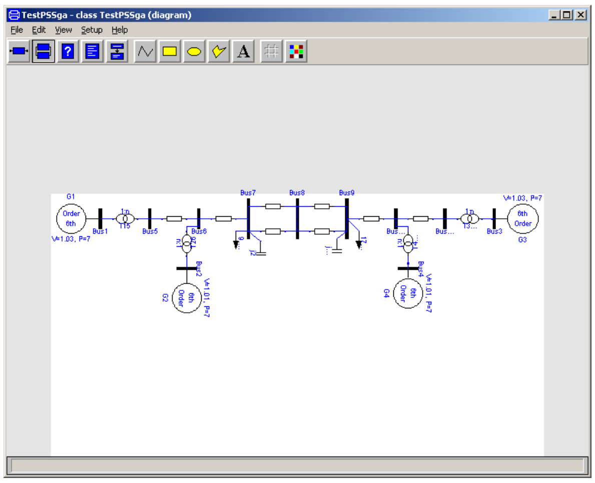


Fig.3 Power system model

6. Demonstration Example

In this section, we will describe the implementation of a GA to a Modelica library called ObjectStab by using 2 different objective functions as described in section 4. Considering the procedure chart in Fig.2, we need to follow 5 steps. It should be noted that only step 5 is different when changing the objective function of a GA. This is due to the manner in which a GA uses the fitness function to evaluate the goodness of solutions that provides greater flexibility of using GA to realize many difficult problems.

In order to follow the procedure in Fig.2, first task is to build a power system model in Dymola. By using the objective function in method 1, we need 3 output variables which are the speed difference of generator 1 and 2, the speed difference of generator 1 and 3, and the speed difference of generator 1 and 4. It should be noted that this is because generator 1 is taken as a reference, hence, the speed difference of generators within the same area is represented as the local mode and the speed difference of generator with different area is represented as the inter-area mode.

Next, we build 3 models in Simulink. Model 1 as shown in Fig.4 can be constructed by drag and drop a DymolaBlock which can be found in Simulink's library browser to a Simulink model. Model name and its path of the Modelica model must be specified in the DymolaBlock in order to point the location of a created Modelica model. It is possible that users can modify a Modelica model directly by using the editing command in the DymolaBlock or compiling a Modelica model by using compiling command. In order to make a Modelica model useful in Simulink and a GA, we will declare external outputs of a Modelica model. These outputs are used for evaluating the fitness value in a GA. The following script is an example of external output declaration in a Modelica model.

```
class TestPSSga
   extends ObjectStab.Examples.Kundur126.linefault;
   Real w1, w2, w3, w4;
   output Real w12;
   output Real w13;
   output Real w14;
equation
   w1 = G1.w;
   w2 = G2.w;
   w3 = G3.w;
   w4 = G4.w;
   w12 = w1 - w2;
   w13 = w1 - w3;
   w14 = w1 - w4;
end TestPSSga
```

After compiling the model, the declared outputs will appear in the DymolaBlock. These outputs can be connected with other Simulink blocks. Now, we can covert a Modelica model to a compiled MEX S-function file by using the following MATLAB commands

```
dymcomp;
[p, x0, pnames, x0names, inputnames, outputnames] = loaddsin;
```

The first command line is used to generate a compiled MEX S-function file (dll file). The second command line is used to load values such as parameters, initial conditions and their names from *dsin.txt* which are necessary for input parameters to S-function block. GA will change parameters *p* every iterations according to the decoded chromosome.

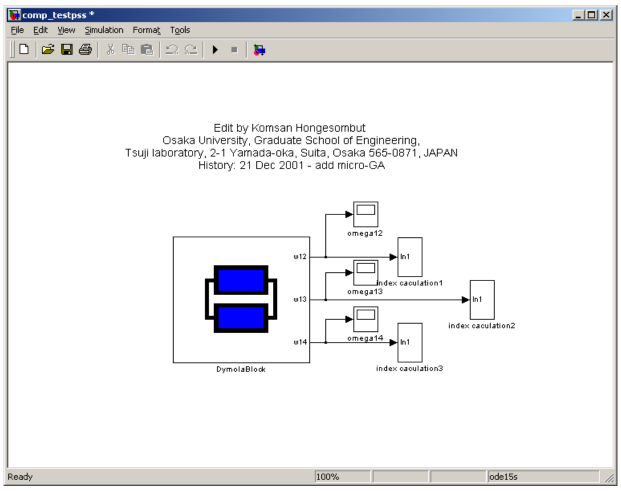

Edit by Komsan Hongesombut
Osaka University, Graduate School of Engineering,
Tsuji laboratory, 2-1 Yamada-oka, Suita, Osaka 565-0871, JAPAN
History: 21 Dec 2001 - add micro-GA

Fig.4 Model 1 in Simulink

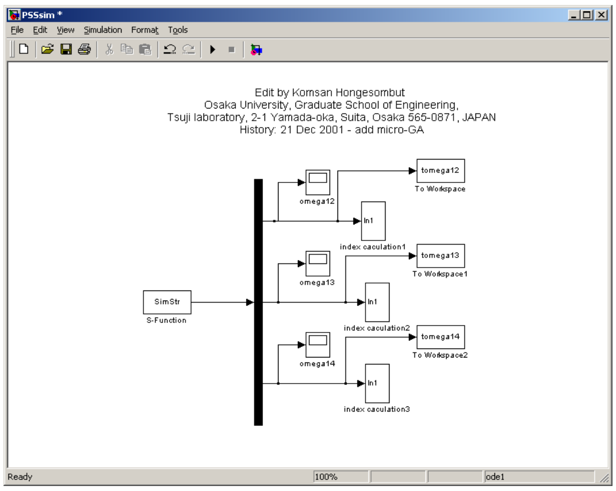

Edit by Komsan Hongesombut
Osaka University, Graduate School of Engineering,
Tsuji laboratory, 2-1 Yamada-oka, Suita, Osaka 565-0871, JAPAN
History: 21 Dec 2001 - add micro-GA

Fig.5 Model 2 in Simulink

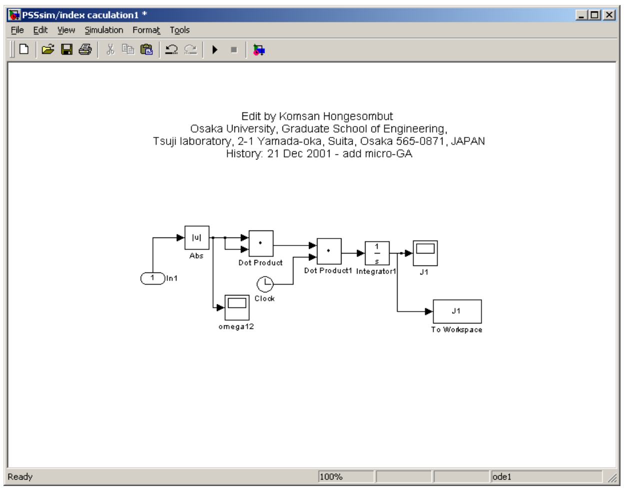
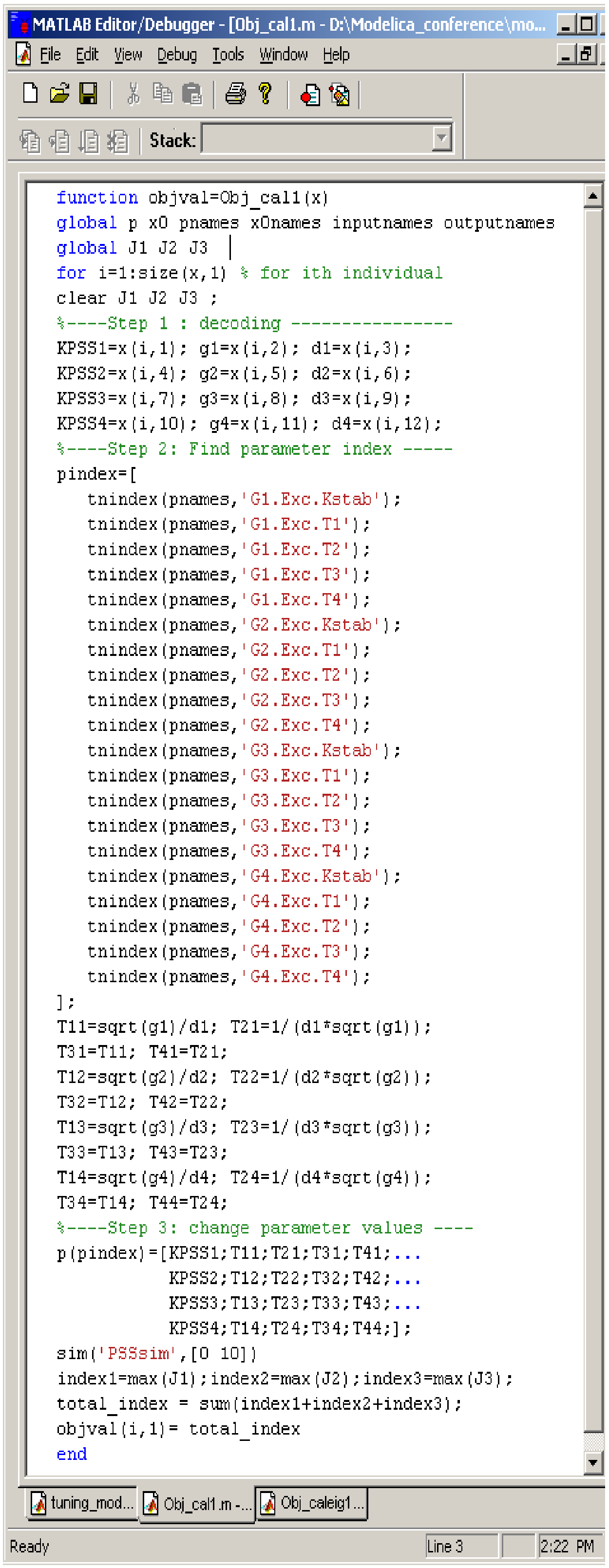
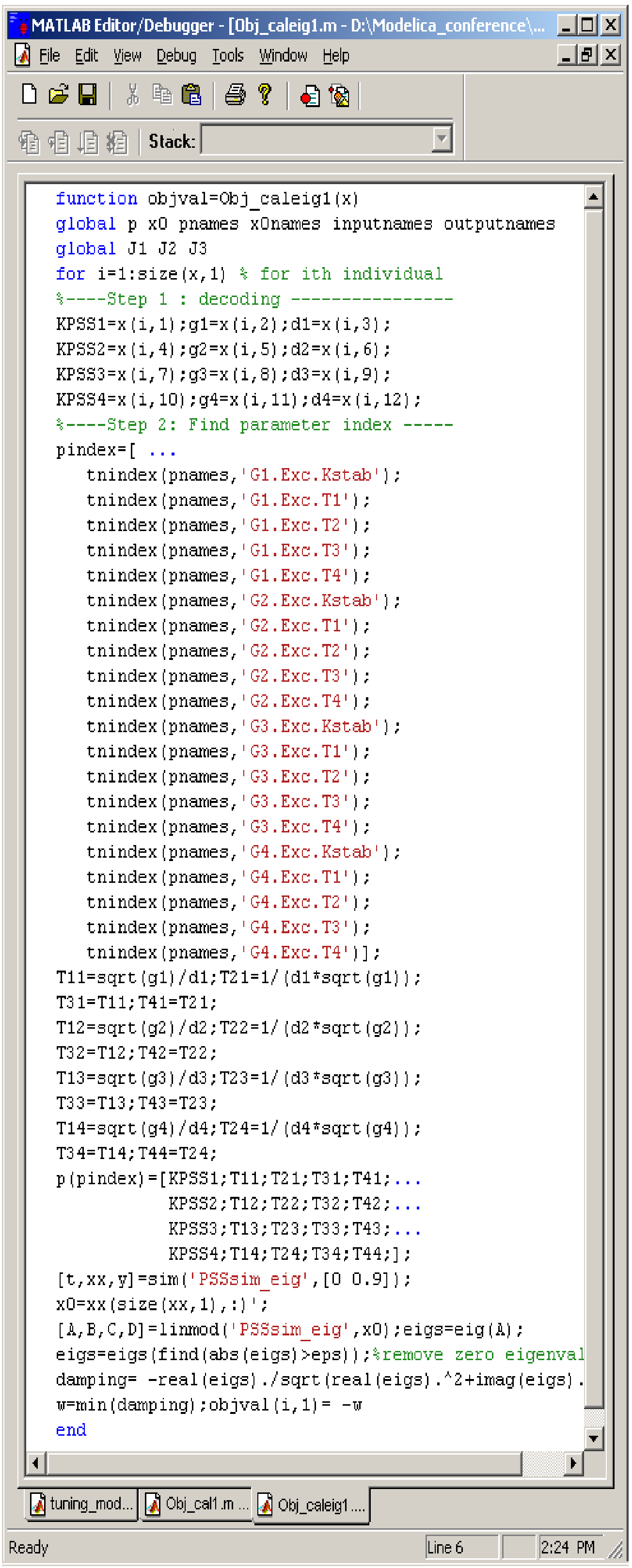

Edit by Komsan Hongesombut
Osaka University, Graduate School of Engineering,
Tsuji laboratory, 2-1 Yamada-oka, Suita, Osaka 565-0871, JAPAN
History: 21 Dec 2001 - add micro-GA

Fig.6 Model 3 in Simulink


MATLAB Editor/Debugger - [Obj_cal1.m - D:\Modelica_conference\mo...
File Edit View Debug Tools Window Help
Stack:
tuning_mod...
Obj_cal1.m -...
Obj_caleig1...
Ready
Line 3
2:22 PM

```
function objval=Obj_cal1(x)
global p x0 pnames x0names inputnames outputnames
global J1 J2 J3
for i=1:size(x,1) % for ith individual
clear J1 J2 J3 ;
%----Step 1 : decoding ----------------
KPSS1=x(i,1); g1=x(i,2); d1=x(i,3);
KPSS2=x(i,4); g2=x(i,5); d2=x(i,6);
KPSS3=x(i,7); g3=x(i,8); d3=x(i,9);
KPSS4=x(i,10); g4=x(i,11); d4=x(i,12);
%----Step 2: Find parameter index -----
pindex=[
   tnindex(pnames,'G1.Exc.Kstab');
   tnindex(pnames,'G1.Exc.T1');
   tnindex(pnames,'G1.Exc.T2');
   tnindex(pnames,'G1.Exc.T3');
   tnindex(pnames,'G1.Exc.T4');
   tnindex(pnames,'G2.Exc.Kstab');
   tnindex(pnames,'G2.Exc.T1');
   tnindex(pnames,'G2.Exc.T2');
   tnindex(pnames,'G2.Exc.T3');
   tnindex(pnames,'G2.Exc.T4');
   tnindex(pnames,'G3.Exc.Kstab');
   tnindex(pnames,'G3.Exc.T1');
   tnindex(pnames,'G3.Exc.T2');
   tnindex(pnames,'G3.Exc.T3');
   tnindex(pnames,'G3.Exc.T4');
   tnindex(pnames,'G4.Exc.Kstab');
   tnindex(pnames,'G4.Exc.T1');
   tnindex(pnames,'G4.Exc.T2');
   tnindex(pnames,'G4.Exc.T3');
   tnindex(pnames,'G4.Exc.T4');
];
T11=sqrt(g1)/d1; T21=1/(d1*sqrt(g1));
T31=T11; T41=T21;
T12=sqrt(g2)/d2; T22=1/(d2*sqrt(g2));
T32=T12; T42=T22;
T13=sqrt(g3)/d3; T23=1/(d3*sqrt(g3));
T33=T13; T43=T23;
T14=sqrt(g4)/d4; T24=1/(d4*sqrt(g4));
T34=T14; T44=T24;
%----Step 3: change parameter values ----
p(pindex)=[KPSS1;T11;T21;T31;T41;...
           KPSS2;T12;T22;T32;T42;...
           KPSS3;T13;T23;T33;T43;...
           KPSS4;T14;T24;T34;T44;];
sim('PSSsim',[0 10])
index1=max(J1);index2=max(J2);index3=max(J3);
total_index = sum(index1+index2+index3);
objval(i,1)= total_index
end
```

(a) Objective function by method 1


MATLAB Editor/Debugger - [Obj_caleig1.m - D:\Modelica_conference\...
File Edit View Debug Tools Window Help
Stack:
tuning_mod...
Obj_cal1.m ...
Obj_caleig1....
Ready
Line 6
2:24 PM

```
function objval=Obj_caleig1(x)
global p x0 pnames x0names inputnames outputnames
global J1 J2 J3
for i=1:size(x,1) % for ith individual
%----Step 1 : decoding ----------------
KPSS1=x(i,1);g1=x(i,2);d1=x(i,3);
KPSS2=x(i,4);g2=x(i,5);d2=x(i,6);
KPSS3=x(i,7);g3=x(i,8);d3=x(i,9);
KPSS4=x(i,10);g4=x(i,11);d4=x(i,12);
%----Step 2: Find parameter index -----
pindex=[ ...
   tnindex(pnames,'G1.Exc.Kstab');
   tnindex(pnames,'G1.Exc.T1');
   tnindex(pnames,'G1.Exc.T2');
   tnindex(pnames,'G1.Exc.T3');
   tnindex(pnames,'G1.Exc.T4');
   tnindex(pnames,'G2.Exc.Kstab');
   tnindex(pnames,'G2.Exc.T1');
   tnindex(pnames,'G2.Exc.T2');
   tnindex(pnames,'G2.Exc.T3');
   tnindex(pnames,'G2.Exc.T4');
   tnindex(pnames,'G3.Exc.Kstab');
   tnindex(pnames,'G3.Exc.T1');
   tnindex(pnames,'G3.Exc.T2');
   tnindex(pnames,'G3.Exc.T3');
   tnindex(pnames,'G3.Exc.T4');
   tnindex(pnames,'G4.Exc.Kstab');
   tnindex(pnames,'G4.Exc.T1');
   tnindex(pnames,'G4.Exc.T2');
   tnindex(pnames,'G4.Exc.T3');
   tnindex(pnames,'G4.Exc.T4')];
T11=sqrt(g1)/d1;T21=1/(d1*sqrt(g1));
T31=T11;T41=T21;
T12=sqrt(g2)/d2;T22=1/(d2*sqrt(g2));
T32=T12;T42=T22;
T13=sqrt(g3)/d3;T23=1/(d3*sqrt(g3));
T33=T13;T43=T23;
T14=sqrt(g4)/d4;T24=1/(d4*sqrt(g4));
T34=T14;T44=T24;
p(pindex)=[KPSS1;T11;T21;T31;T41;...
           KPSS2;T12;T22;T32;T42;...
           KPSS3;T13;T23;T33;T43;...
           KPSS4;T14;T24;T34;T44;];
[t,xx,y]=sim('PSSsim_eig',[0 0.9]);
x0=xx(size(xx,1),:)';
[A,B,C,D]=linmod('PSSsim_eig',x0);eigs=eig(A);
eigs=eigs(find(abs(eigs)>eps));%remove zero eigenval
damping= -real(eigs)./sqrt(real(eigs).^2+imag(eigs).
w=min(damping);objval(i,1)= -w
end
```

(b) Objective function by method 2

Fig.7 Comparison of two objective function used by a GA

After we get a compiled MEX S-function file which has a default name SimStr.dll, as stated earlier, we need to calculate $\int|\Delta\omega|^2 t \cdot dt$ for each generator speed deviation. Model 2 shown in Fig.5 is served for this function where model 3 shown in Fig.6 is a sub-system for calculation $\int|\Delta\omega|^2 t \cdot dt$ of each speed signal. The summation of 3 speed signals become the objective function of a GA by using the method 1. Particularly useful in conjunction with a GA is the way to write the objective function. It is worthwhile to discuss the construction of the objective function. In Fig.7, it shows the comparison of two objective functions used in this study. The meaning behind each style is

Method 1:
1. Decode the chromosome of a GA.
2. Find the index of parameters which correspond to the tuning parameters in a Modelica model. The syntax of this command is

```
pindex = tnindex(pnames, ‘parameter name’)
```

where pnames is obtained from *loaddsin* command

3. Replace current parameters with new parameters obtained by a GA.
4. Run the Simulink model with *sim* command. Simulink will run the model 2 and save the index calculation of each signal when the simulation is complete.
5. Calculate the fitness value by summing 3 signals which each signal is calculated by subsystem model 3.

Method 2:
1. Decode the chromosome of a GA.
2. Find the index of parameters which correspond to the tuning parameters in a Modelica model.
3. Replace current parameters with new parameters obtained by a GA.
4. Run the Simulink with *sim* command in order to find good initial condition *x0.*
5. When the initial values are obtained, we can then proceed to use the MATLAB *linmod* function to determine the [A, B, C, D] matrices of the small-signal model of the nonlinear system about the chosen steady-state operating point. The syntax of the linearization command is as follows:

```
[A, B, C, D] = linmod(‘model name’, x0)
```

It should be noted that when calculating the eigenvalues, it is not necessary to have an input, but there should be at least one output of a Modelica model.
6. Calculate the fitness value by (8).

7. Simulation Results

A GA is applied to solve the problem of simultaneous tuning by using 2 different objective functions. In this study, routines from GEATbx were used with bounds for PSS parameters shown in Table 1. The implementation of a GA in this work used real encoding chromosome, a population size 30, maximum generation 50, a uniform crossover rate of 0.9 and a uniform mutation rate of 0.01. The approach also adopted an elitist strategy that copied the best string found in the current generation to the next generation. Selection was performed by using the tournament selection with tournament size of 2. After executing a GA, the final result as shown in Table 2 were obtained. Fig.8 and 9 show the screen outputs of a GA by using the objective function by method 1 and method 2 respectively.

Table 1 Bounds for PSS parameters

PSS parameter	value
K_{min}	0
K_{max}	20
γ_{min}	0.1
γ_{max}	10
δ_{min}	1
δ_{max}	10

Table 2 Final result obtained by a GA

Method 1	K	$T_1 = T_3$	$T_2 = T_4$
PSS1	20.000	0.331	0.139
PSS2	20.000	0.107	0.291
PSS3	17.791	0.127	0.153
PSS4	18.319	0.201	0.055
Method 2	K	$T_1 = T_3$	$T_2 = T_4$
PSS1	19.175	1.583	0.632
PSS2	20.000	0.161	0.184
PSS3	14.209	0.198	0.239
PSS4	7.513	0.051	0.218

To demonstrate the effectiveness of the resulting controller obtained by using 2 objective functions, nonlinear simulation and plot of close-loop eigenvlaues were performed. In nonlinear simulations of Fig.10 to 12, the responses of generator speed deviation for local and inter-area modes confirm the effectiveness of the results obtained by a GA. It should be noted that the method 1 gives better result than the method 2 when using time domain-based performance index. The system is well damped and is stabilized in less than 5 seconds.

Fig. 13 to 14 show the plot of dominant eigenvalues of the closed-loop system. It can be observed that using PSS parameters obtained by both methods, the system is sufficiently damped with all modes of the

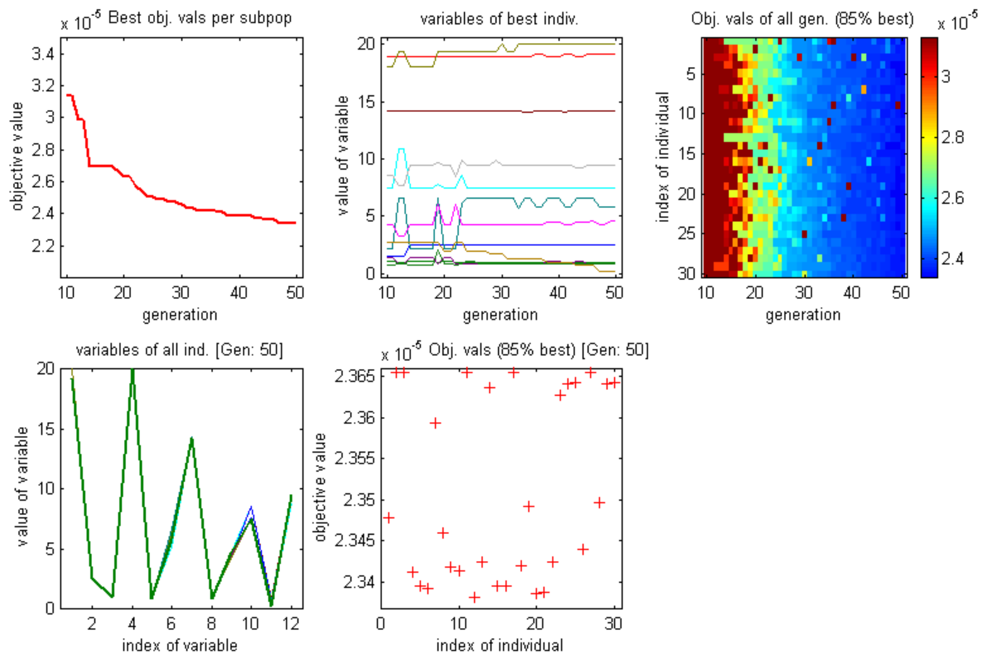


Fig.8 Screen output from a GA by using the objective function in method 1

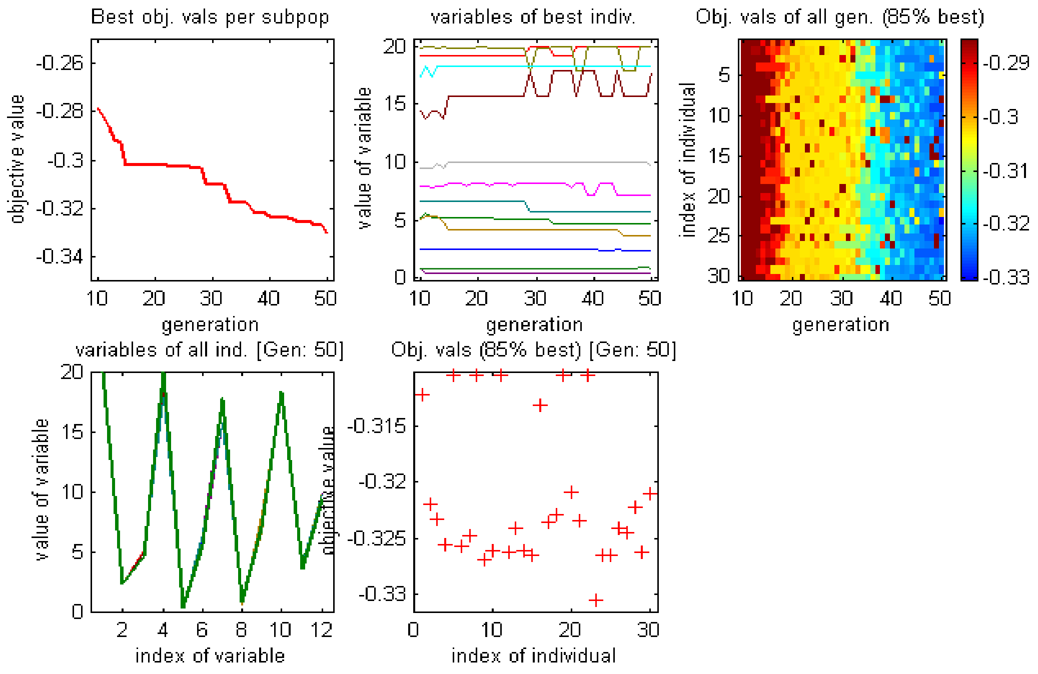


Fig.9 Screen output from a GA by using the objective function in method 2

system having the minimum damping greater 5% which is a typical requirement in PSS tuning. It is also found that the method 2 gives better result than the method 1 in case of using eigenvalue-based performance index.

It is become clear that using different GA objective function, the final result may be quite different. In addition, GA is a time consuming search procedure. Thus, GA is not generally used for problems easily optimized.

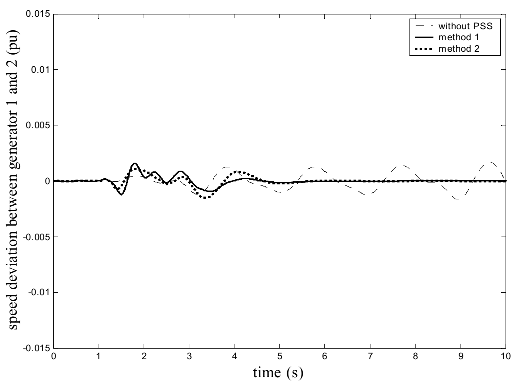


Fig.10 Speed deviation of generator 1 and 2

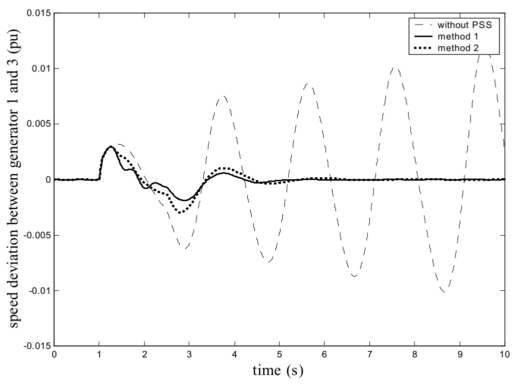


Fig.11 Speed deviation of generator 1 and 3

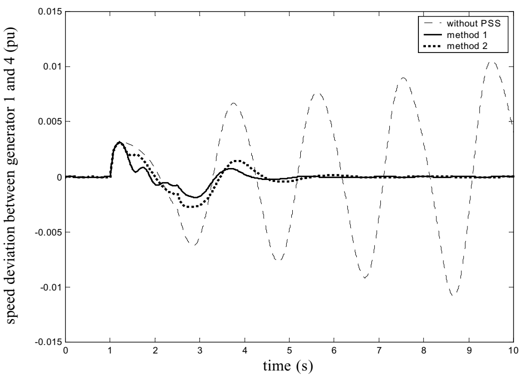


Fig.12 Speed deviation of generator 1 and 4

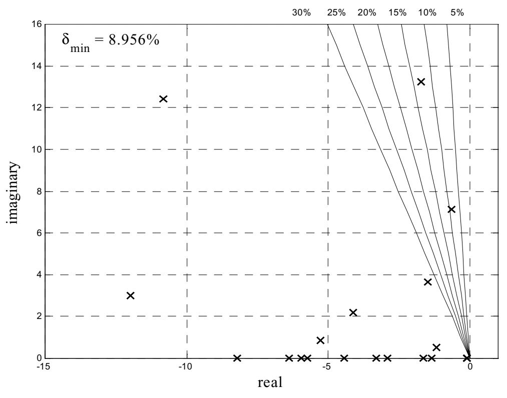


Fig.13 Closed-loop eigenvalues obtained by method 1

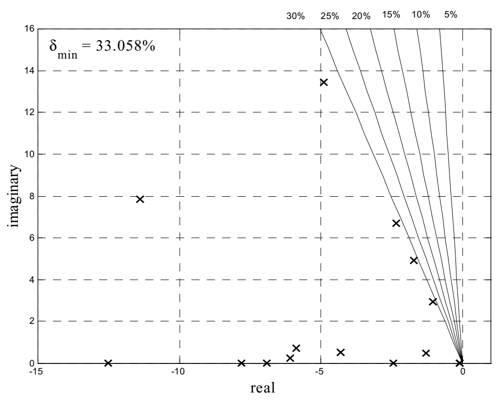


Fig.14 Closed-loop eigenvalues obtained by method 2

8. Conclusions

This paper deals with the incorporated use of a Modelica library called ObjectStab and a GA and application of a GA for simultaneous tuning of power system stabilizers in a multimahcine power system. The power system modeling can be realized by using ObjectStab where the behavior of dynamic systems can be expressed by using advance features of Modelica language for detailed physical modeling. We also showed how to link a GA and a Modelica model by using the Simulink interface of the Dymola. We showed the flexibility of optimization by a GA with two different objective functions without modifying the original Modelica model. Given a suitable objective function, the final solution will satisfy the required controller performance. It is important to point that the idea does not limit only the applications to power systems as shown in an example of this paper, but also other Modelica users can adapt this idea to their own works.

9. Acknowledgement

We gratefully acknowledge helpful discussions with Dr. Mats Larsson from ABB Corporate Research Ltd., Switzerland.

10. References

[1] M. Larsson, "ObjectStab - a Modelica library for power system stability studies", Proc. of the 2000 Modelica Workshop.

[2] Dymola*, Dynamic Modeling Laboratory*, Dynasim 2001.

[3] M. M. Tiller, *Introduction to Physical Modeling with Modelica*, Kluwer Academic Publishers, Massachusetts 2001.

[4] D. Hanselman and B. Littlefield, *Mastering Matlab 6*, Prentice Hall, New Jersey 2001.
[5] J. B. Dabney and T. L. Harman, *Mastering Simulink 4,* Prentice Hall, New Jersey 2001.
[6] H. Pohlheim, *Genetic and Evolutionary Algorithm Toolbox for use with MATLAB*, Publication on internet web at http://www.geatbx.com.
[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, New York, 1989.
[8] P. Kundur, *Power system stability and control* McGrawHill, New York 1993.

Komsan Hongesombut received his B.Eng.(first class honors) and M.Eng. degrees from the Department of Electrical Engineering, King Mongkut's Institute of Technology Ladkrabang, Thailand in 1997 and 1999 respectively. He is currently a Ph.D student at Osaka University, Japan. His research interests include the applications of intelligent techniques to power systems. He is a student member of the Institute of Electrical Engineers of Japan, IEE, and IEEE.

Yasunori Mitani received his B.Sc., M.Sc., and Dr. of Engineering degrees in electrical engineering from Osaka University, Japan in 1981, 1983, and 1986 respectively. He joined the Department of Electrical Engineering of the same university in 1990. He is currently Associate Professor. His research interests are in the areas of analysis and control of power systems. He is a member of the Institute of Electrical Engineers of Japan, the Institute of Systems, Control and Information Engineers of Japan, and the IEEE.

Kiichiro Tsuji received his B.Sc and M.Sc. degrees in electrical engineering from Osaka University, Japan, in 1966 and 1968, respectively, and his Ph.D in systems engineering from Case Western Reserve University, Cleveland, Ohio in 1973. In 1973 he jointed the Department of Electrical Engineering, Osaka University, and is currently Professor. His research interests are in the areas of analysis, planning, and evaluation of energy systems, including electrical power systems. He is a member of the Institute of Electrical Engineers of Japan, the Japan Society of Energy and Resources, the Society of Instrument and Control Engineers, the Institute of Systems, Control and Information Engineers, and the IEEE.