Using Fault Augmented Modelica Models for Diagnostics

Raj Minhas Johan de Kleer Ion Matei Bhaskar Saha
Bill Janssen Daniel G. Bobrow Tolga Kurtoglu
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
contact email: dekleer@parc.com

Abstract

We propose a model-based diagnosis framework in which Modelica models of faulted behavior are used in combination with a Bayesian approach. The fault augmented models are automatically generated through a process developed as part of our Fault Augmented Model Extension (FAME) work. Fault diagnosis using a Bayesian approach is based on computing a set of probability density functions, a process that is usually intractable for any reasonably complex system. We use Approximate Bayesian Computation (ABC) to bound the numerical and computational complexity. The basic idea is to use fault augmented Modelica models to create probability distributions of possible outcomes and then compare those distributions against actual observations to perform parameter estimation. The detection of faults is treated as a model selection problem and the inference of their severity levels is treated as parameter estimation. The diagnostic precision of this approach is evaluated on a Modelica vehicle drive line model.

Keywords: fault models; diagnosis; machine learning; model translation;bayesian methods

1 Introduction

Modelica [6] is an object-oriented, declarative, multi-domain modeling language for component-oriented modeling of complex systems. It is used to execute numerical simulations to determine the behavior of a system. This approach frees the designer to efficiently explore a wide set of designs to see which meets customer requirements, without needing to physically construct the systems.

In addition to predicting behaviors through numerical simulations of Modelica models, we propose to use the same simulation models for diagnosis. Modelica's focus on simulation would seem to make it a poor choice for diagnosis. After all, diagnosis is the inverse of simulation. Simulation predicts the behavior of a system given a (correct) model. Diagnosis must infer how the model has changed (i.e., faulted) from observed behavior.

Most model-based diagnosis algorithms [4, 10] perform inference on declarative models. Although Modelica supports the writing of declarative models, too many Modelica models (including many in the MSL) contain imperative constructs making direct application of existing model-based diagnosis algorithms problematic. RODON [2] is a Modelica inspired approach to modeling, but Modelica models first have to be re-written by hand in qualitative declarative form. We know of no system identification or FDI technique that applies for DAE models with boolean constraints (as Modelica models translate into). Our approach, on the other hand, applies on Modelica models directly no matter what types of constraints they contain.

This paper presents a framework for model-based diagnosis in which Modelica tools play a fundamental inference role in a Bayesian approach. Numerical simulations of Modelica models are used to build statistical models for the behavior of a system for all of its fault-operating modes; statistical models that help determine a diagnosis solution based on observations.

The simulation of a system under different fault-operating modes is enabled by fault-augmented Modelica libraries. As part of our Fault Augmented Model Extension (FAME) work, we have developed an approach [5] for automatic model construction of Modelica fault models. Our model fault-augmentation approach is based on analyzing Modelica libraries for fault susceptibility and on modifying them to enable simulation of faulty components. FAME provides the set of possible faulted behaviors for each component (e.g., a resistor might be open, shorted, or resistance shifted by some undetermined amount). FAME can use as input statistical models from reliability analysis that determine the fault activations and the amount

of change in the values of the parameters.

Consider a grossly simplified vehicle model illustrated in the Modelica model of Figure 1. The brake may be working normally (`Nominal` state) or may be having friction related faults (`Slipping` or `Sticking`). The severity of these fault modes is modeled in Modelica by a parameter called `amount`$\in [0,1]$ where a value of 0 indicates no fault (i.e., `Nominal` state) and a value of 1 indicates total failure. The part will likely be unusable (and hence considered faulty) for values far less than 1.

When the brake absolute angular velocity ω is not zero, the frictional torque applied by the brake is a function of a velocity dependent friction coefficient $\mu(\omega)$, the normal force f_n, and of a geometry constant c_{geo}, which takes into account the geometry of the device and the assumptions on the friction distributions:

$$\tau = c_{geo} \times \mu(\omega) \times f_n.$$

Our approach can be used to estimate such parameters in addition to performing diagnostics. For this exercise, we assume that the only unknown parameter is the `amount` but we could just as easily estimate the other parameters such as c_{geo} and f_n from the observed data.

2 FAME approach

We have developed faultable models for the models in the Modelica Standard Library. A detailed description of FAME can be found in [5], while [8] describes how it can be used in a design-tool chain to perform reliability analysis. Here we summarize the essential details of FAME for the purpose of diagnosis. Every model class definition which contains faults is replaced with a new class definition, a Modelica model class subsuming the original model class and adding declarative behavior to allow simulation of the faults.

An encapsulated enumerated type is defined, listing the various fault modes of the class, along with the `Nominal` mode. A discrete mode parameter of this new type is introduced, defining the mode in which an instance of the class is operating. An if-equation similar to Figure 2 is added, so that each operating mode can define its own dynamics.

The set of equations which apply in each fault mode is expressed in the appropriate branch of this if-equation. The process also flattens the superclasses of the model into the rewritten class, and introduces two new externally visible parameters, `mode` and

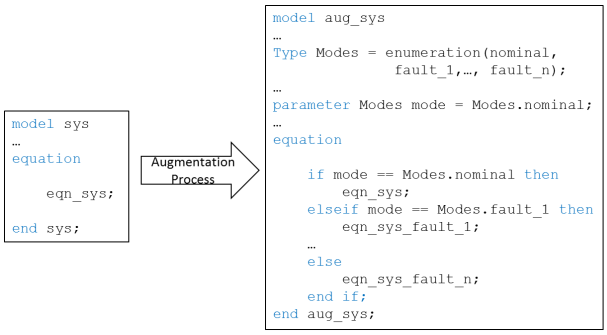


Figure 2: Alternative dynamics are enabled for each operating mode.

`amount`, as well as an enumerated type giving the possible faults for this component, `Modes`.

Modelica models are augmented with two types of faults: *power faults* and *parametric faults*. Power faults model loss of power at the connection points and is implemented through the addition of new components abstractly called "dampers" that implement this behavior. We show below how the `Stick` fault is implemented at `flange_a` of the `Brake` component:

```
model Brake
...
Modelica.Mechanics.Rotational.
            Interfaces.Flange_a flange_a;
FAME.Dampers.RotationalWithConnectEquations
   _famefault_flange_a(amount=0.0);
...
end Brake

model RotationalWithConnectEquations
  input Real amount(min=0.0, max=1.0);
  encapsulated type Modes = enumeration(
      Nominal,
      Stick,
      Broken);
  parameter Modes mode = Modes.Nominal;
  Modelica.Mechanics.Rotational.
                Interfaces.Flange_a port_a;
  Modelica.Mechanics.Rotational.
                Interfaces.Flange_a port_b;
equation
   ...
  elseif mode == Modes.Stick then
    port_b.tau + port_a.tau =
    (1/max(Constants.verySmall,1-amount)-1)
    *der(port_a.phi);
    port_b.phi = port_a.phi;
  else
    ...
  end if;

end RotationalWithConnectEquations;
```

We note that a new component called `_famefault_flange_a` was added to the `Brake` model; component that implements the behavior corresponding to the `Stick` fault.

DOI 10.3384/ECP14096437

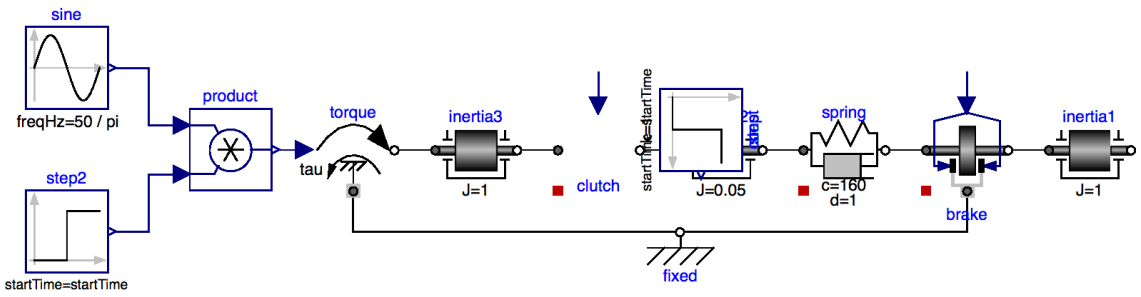


Figure 1: Simple drive line.

Similarly, parametric faults are handled by introducing a new component that defines the pattern of change for the parameter to which this component is associated. For example in the case of the `Brake` model, `_famefault_mue_pos` is an instance of the newly added `_famefaults_mue_pos` component that corresponds to the `mue_pos` parameter:

```
model Brake
 model _famefaults_mue_pos
   extends FAME.Parametric.
     BaseParametricFault(amount=0.0);
   type Modes = enumeration(
        Nominal,
        Slip);
   parameter Modes mode=Modes.Nominal;
   equation
  if mode==Modes.Slip then
   y = u*{{1,0},{0,1-amount}};
  else
   y = u;
  end if;
 end _famefaults_mue_pos;
...
 _famefaults_mue_pos _famefault_mue_pos
    (u=mue_pos,redeclare type
    ParamType = Real[size(mue_pos,1),2]);
...
end Brake;
```

References to the original parameter are replaced with an expression which references to this new variable. For example, the parameter `mue_pos` is replaced in the fault-augmented version of the `Brake` by `_famefault_mue_pos.y`.

We consider several other fault modes. Consider the simple spring-damper system of Figure 3. Figure 4

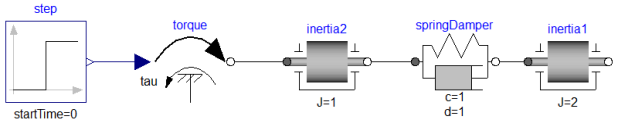


Figure 3: Spring Damper example.

shows three simulation results for the `Nominal`, `Stick` and `Broken` modes for `inertia1`. The underlying intuition of our approach is to pre-compute many simulations under many fault scenarios and perform on-line diagnosis by identifying the pre-computed simulation results which best matches the observation. In the remainder of this paper we describe how this intuition has been implemented.

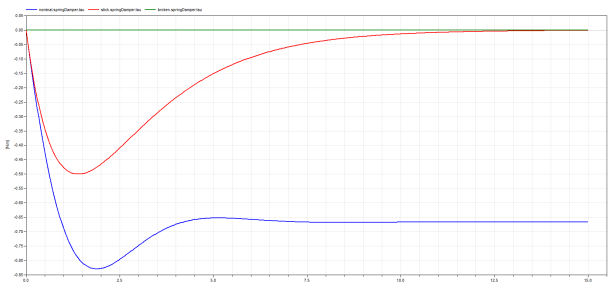


Figure 4: `Nominal`, `Stick` and `Broken` operating modes for the spring-damper system

3 Fault detection in the drivetrain system

We demonstrate our proposed approach by detecting faults in the drivetrain system shown in Figure 1. We model five failure modes: brake slipping, brake sticking, clutch slipping, clutch sticking, and spring sticking. The failure mode of a stuck brake is modeled in the FAME library as a dynamic damper component that adds damping equal to the `amount`[1]. This has the effect of increasing the relative friction in the flanges which results in a loss of power. The failure mode of a slipping brake is modeled as a decrease in the velocity dependent friction coefficient by an amount equal to the severity of the fault:

$$\mu(\omega) \mapsto \mu(\omega) \times (1 - amount).$$

[1]Under normal operating conditions, the `amount`$= 0$ so the system does not experience this additional damping.

This has the effect of reducing the friction torque[2]. The other failure modes are modeled in a similar manner. Our aim is to infer from observed data whether the system is exhibiting a fault mode — if so, we also need to infer the severity of the fault (i.e., the value of `amount`). We achieve this by using ideas developed for Approximate Bayesian Computation (ABC).

4 Approximate Bayesian Computation

Modelica models are formal representations of (hybrid) differential equations (DAEs). Existing results on fault diagnosis of DAEs are not able to cope with such mathematical models in their generality. They either focus on the structural aspect of the system, ignoring its dynamics [7], or they make simplifying assumptions, such as linearity, on the DAE model [3]. Therefore more practical approaches for fault diagnosis must be employed.

In this section, we give an overview of the ABC method, based on material from [9, 12, 1] — see those references for more details. The typical tasks are to estimate unknown model parameters and to do model selection. Let M be a model parameterized by some parameters Θ whose joint prior density is $\pi(\Theta)$. Given data d generated by the model M, we are interested in estimating the posterior probability $\pi(\Theta|d)$. From Bayes rule, we know that

$$\pi(\Theta|d) \propto f(d|\Theta)\pi(\Theta),$$

where $f(d|\Theta)$ is the likelihood of the data given the parameters. The prior probability of the parameters is known so we need to compute the likelihood of the data in order to estimate the posterior probability of the parameters. A similar approach can be used for model selection. Let M_1 and M_2 be two models and we would like to infer which of them is more likely to have generated the given data d. Using the Bayesian analysis framework, we compute the Bayes factor B_{12} to summarize the evidence provided by the data for model M_1 over model M_2:

$$B_{12} = \frac{P(M_1|d)/P(M_2|d)}{P(M_1)/P(M_2)},$$

where $P(M_i)$ is the prior probability of model M_i and $P(M_i|d)$ is the posterior probability of the model M_i given the data d. A value of B_{12} between 1 and 3 suggests weak evidence in favor of M_1, a value between 3 and 20 suggests positive evidence, a value between 20 and 150 suggests strong evidence, and a value greater than 150 suggests very strong evidence. The prior probabilities of the models don't depend on the given data and can be pre-computed. So computing the Bayes factor comes down to computing the ratio of the posterior probabilities of the models given the data:

$$B_{12} \propto \frac{P(M_1|d)}{P(M_2|d)} \propto \frac{f(d|\Theta_1)}{f(d|\Theta_2)}.$$

For any reasonably complex model, the likelihood computation is intractable so ABC approaches like rejection are used to approximate it. In order to simplify the problem computationally, it is common to define a function $f_s : \mathbb{R}^n \to \mathbb{R}^m$ that maps the given data $d \in \mathbb{R}^n$ to some representative statistic $s \in \mathbb{R}^m$ where $m \ll n$. Then we define a distance metric $dist_s : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ to measure closeness of two sets of statistics. Let $\hat{d}_{M_i}$ be simulated data generated from a model M_i.

4.1 Rejection Technique

If $dist_s(f_s(d), f_s(\hat{d}_{M_i})) < \varepsilon$ for some threshold ε then the simulated data is accepted for the given observed data d — otherwise, it is rejected. Now assume that we generate N data sets simulated from the model M_i as follows. For each of the N iterations, draw a parameter vector Θ_i from the prior distribution $\pi(\Theta_i)$ and simulate data d_s^i from M_i. Assume that $\bar{N}$ of those simulated data sets are accepted given a threshold ε. Then we can approximate $P(M_i|d)$ as $\frac{\bar{N}}{N}$. The distribution of the Θ_i for each of the accepted iteration approaches $\pi(\Theta_i|d)$. The approximated values approach the true values asymptotically as $N \to \infty$ and $\varepsilon \to 0$ if the statistics are sufficient to describe the data. The downside of this approach is that if ε is small then N needs to be high in order to achieve a reasonable approximation. In other words, it can be very computationally expensive.

4.2 Classification Technique (alternative approach)

For model selection, rather than computing the ratio of posterior probabilities, we could use a classification approach instead. Here we treat the model indicator i as the response variable and the summary statistics as the dependent variables. Any standard classification technique such as multinomial logistic regression, random forest, neural network etc. can be used

[2] Again, under normal operating condition, the `amount` $= 0$ so the system does not experience this loss in torque.

DOI 10.3384/ECP14096437

to train a model based on the simulation data.[3] The trained model is evaluated on the statistics $f_s(d)$ of the observed data to directly estimate $P(M_i|d)$. This approach may be needed for more complex diagnostic tasks, but as the rejection approach performs so well, we use it in our example.

5 Evaluating the approach

The first step in ABC is to generate a large number of simulations. To constrain the problem appropriately, we assume that the observed output of the system is generated in response to a known input. For example, the response of a dynamical system to a step input (called a *s*tep response) is typically used to reason about the system behavior and may be appropriate for diagnosis as well. This choice is part of the feature selection and needs to be made at the time of diagnostic design. For each of the five fault modes, we generate 10,000 step-response simulations[4] — we first sample the `amount` from its prior distribution[5] and then use that value to perform the simulation. This was done by parameterizing the variables in Modelica file — a snippet of the Modelica model is shown below — the string `FAULT_AMOUNT` is replaced with the sampled value before running a simulation.

```
Brake brake(fnmax = 1600,
 _famefault_mue_pos(mode = Modes.Slip,
 amount=FAULT_AMOUNT);
```

The second step in ABC is to compare observed data against the simulated data. For computational reasons, we compare features computed from the simulated and actual data rather than comparing the raw data directly. For this exercise, we evaluate the step response of the drive train system and compute the following features of the absolute angular velocity of the inertial load connected to the brake: (1) mean, (2) maximum, (3) 25^{th} percentile, (4) 50^{th} percentile, (5) 75^{th} percentile, (6) inter-quartile range, and (7) time to go to zero. The values of these seven features can be thought of as a vector of dimension seven. The difference between the observed vector and the simulated vectors is used to compute an estimate of the likelihood that the observed value was generated from the simulated distribution. For more details, please see [11].

In order to evaluate the effectiveness of our approach, we measure the diagnostic accuracy of detection the following 11 faults.

1. *Brake Slipping* fault mode with `amount`$= 0.1$
2. *Brake Slipping* fault mode with `amount`$= 0.25$
3. *Brake Sticking* fault mode with `amount`$= 0.1$
4. *Brake Sticking* fault mode with `amount`$= 0.25$
5. *Clutch Slipping* fault mode with `amount`$= 0.1$
6. *Clutch Slipping* fault mode with `amount`$= 0.25$
7. *Clutch Sticking* fault mode with `amount`$= 0.1$
8. *Clutch Sticking* fault mode with `amount`$= 0.25$
9. *Spring Sticking* fault mode with `amount`$= 0.1$
10. *Spring Sticking* fault mode with `amount`$= 0.25$
11. *Nominal* mode (i.e., with no fault mode or a fault mode with a very small `amount`)

For each of these faults, we infer the fault model that was most likely to have generated it and estimate the `amount`. The analysis was done using the *ABCToolbox* suite [11].

Results

We first show how each model fares against the observed data and then put it all together to generate the final diagnosis.

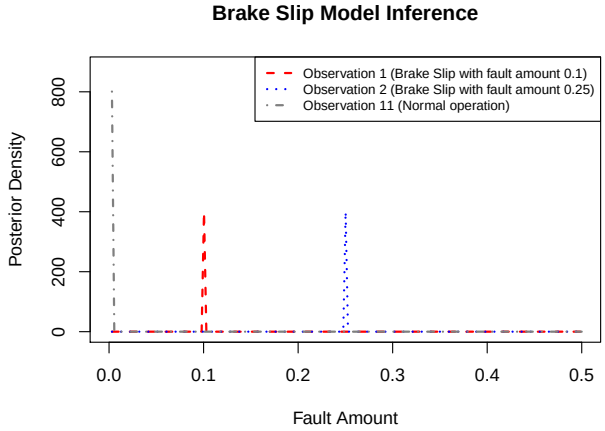
Brake Slipping model

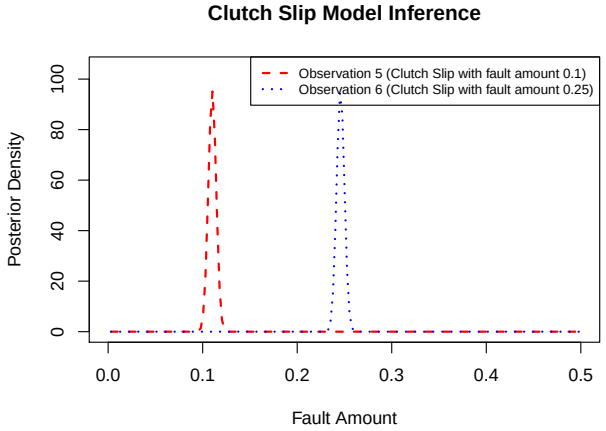
When the eleven faulty behaviors are compared against the simulations from the *Brake Slipping* model, the marginal distribution of the model is nearly zero for all but faults #1, #2 and #11. The graphs below show the posterior density distributions of `amount` for those three faults — as the graph shows, the inference is correct and the estimate of the `amount` is also very accurate. Of course, in order to make the final diagnosis for a fault, the marginal density for this model will need to be compared against the densities for the other models.

[3] Any technique that returns a normalized measure of classification should be usable.

[4] For a more complex model, we may need to generate a higher number of simulations.

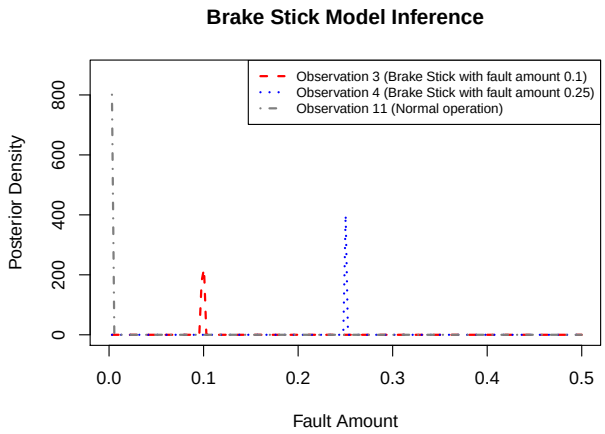
[5] For this example, we sample `amount` from a uniform distribution: `amount` $\sim U(0.003, 0.5)$. In practice, the choice of this distribution will depend on the belief of the designers about the distribution of the fault amount - such a distribution may be learnt from field performance data of similar systems. This is a typical design choice in Bayesian analysis and a non-informative distribution such as a uniform distribution may be used if no other source of information is available.





Brake Sticking model

When the eleven faulty observations are compared against the simulations from the *Brake Sticking* model, the marginal distribution of the model is nearly zero for all but faults #3, #4, and #11. The graph below shows the posterior probability distributions of `amount` for those three faults — again, the inference is correct and the estimate of the `amount` is also very accurate.

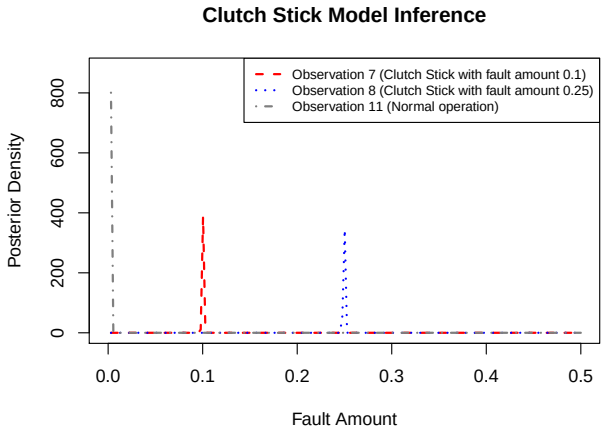


Clutch Slipping model

When the eleven faulty observations are compared against the simulations from the *Clutch Slipping* model, the marginal distribution of the model is nearly zero for all but faults #5, and #6. The graph below shows the posterior probability distributions of `amount` for those two faults — while the distributions are not as peaked as the ones for brakes, the inference is still correct and the mode of the distribution is over the correct value of `amount`.

Clutch Sticking model

When the eleven faulty observations are compared against the simulations from the *Clutch Sticking* model, the marginal distribution of the model is nearly zero for all but faults #3, #7, #8, #9, and #11. In this case, the marginal distribution is non-zero for two faults (#3 and #9) that do not correspond to the clutch sticking failure mode. However, the overall diagnosis still turns out to be correct because the marginal distribution numbers for the correct models are much higher than these numbers (see Section 5). The graph below shows the posterior probability distributions of `amount` for faulty observations #7, #8, and #11 — as in the other cases, the posterior distributions of `amount` have very narrow peaks over the correct values.

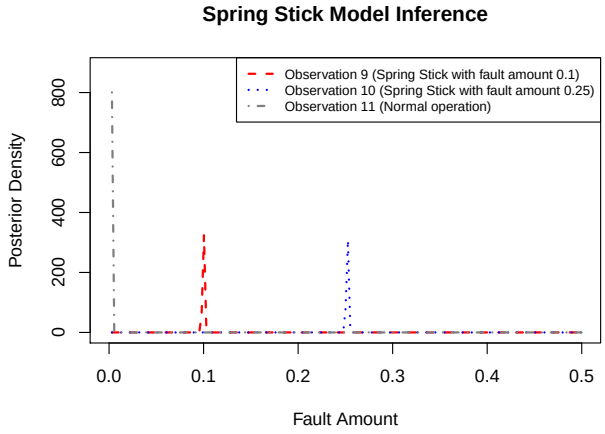


Spring Sticking model

When the eleven faulty observations are compared against the simulations from the *Spring Sticking* model, the marginal distribution of the model is nearly

DOI 10.3384/ECP14096437

zero for all but faults #9, #10, and #11. The graphs below show the posterior probability distributions of `amount` for those faults — again, both the fault mode inferences and the estimates of `amount` are correct.



Final diagnosis

As mentioned previously, all the faulty observations except #3, #9 and #11 are inferred to have a single (and the correct) cause. While fault #11 (which corresponds to the normal operation) has significant ambiguity regarding the cause, the inferred `amount` is always nearly zero in all those cases — this correctly indicates the absence of any fault mode. In other words, fault #11 is correctly associated with normal operation. Fault #3 is inferred to have been generated by either *Brake Sticking* or *Clutch Sticking* failure modes. Similarly, fault #9 is inferred to have been generated by either *Clutch Sticking* or *Spring Sticking* failure modes. So for these two cases, we need to look at the ratios of the respective posterior densities, i.e., compute the Bayes factor in order to complete the diagnosis (see Section 4 for more details).

Fault #3 For this fault, the marginal posterior density of *Brake Sticking* model is 9.8×10^9 while that of *Clutch Sticking* model is 1.7. So the Bayes factor for *Brake Sticking* is $\frac{9.8 \times 10^9}{1.7} = 5.7 \times 10^9$ which is very strong evidence in favor of *Brake Sticking* (i.e., the correct diagnosis).

Fault #9 For this fault, the marginal posterior density of *Clutch Sticking* model is 3.6×10^3 while that of *Spring Sticking* model is 2.9×10^8. So the Bayes factor for *Spring Sticking* is $\frac{2.9 \times 10^8}{3.6 \times 10^3} = 8 \times 10^4$ which is very strong evidence in favor of *Spring Sticking* (i.e., the correct diagnosis).

So the FAME based inference approach is able to make the correct diagnosis for all the faults.

6 Final remarks

This paper has demonstrated that a straight-forward application of machine learning techniques to simulation can be used to accurately diagnose systems. In fact, it can diagnose systems even if some of the (non-faulted) parameters are unknown. This approach has some inherent limitations: (1) the expansion to multiple faults will require exponentially more pre-computed simulations and therefore would only scale to simple systems, (2) active diagnosis will require deriving features for many system variables which may be impractical, (3) the features (i.e. sufficient statistics of the signal) we use are somewhat determined by the requirements of the system and must determined at the outset, and (4) if the system can have a wide variety of exogenous inputs, too many pre-computed simulations will be required in order to diagnose for each possible input stream.

The expansion to multiple faults can be ameliorated somewhat by the fact that the simulations are done offline and can be easily parallelized. This complexity may be further managed if a reasonable assumption can be made about an upper limit on the number of simultaneous faults (thereby reducing the complexity from exponential to polynomial).

7 Acknowledgments

This work was partially sponsored by The Defense Advanced Research Agency (DARPA) Tactical Technology Office (TTO) under the META program. Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] M. G. B. Blum and O. Franscois. Non-linear regression models for approximate bayesian computation. *Statistics and Computing*, 20(20):63–73, 2010.

[2] Peter Bunus and Karin Lunde. Supporting model-based diagnostics with equation-based object oriented languages. In *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools*, Paphos, Cyprus, July 2008.

```
model Spring
   "Linear 1D rotational spring"

  // locally defined classes in Spring
    model _famefaults_flange_a = FAME.Dampers.RotationalWithoutConnectEquationsAasP;
    model _famefaults_flange_b = FAME.Dampers.RotationalWithoutConnectEquationsAasP;
    model _famefaults_c
     extends FAME.Parametric.BaseParametricFaultAmountAsParameter(amount=0.0);

   // locally defined classes in _famefaults_c

    type Modes = enumeration(Nominal, Fatigue);

   // components of _famefaults_c
    parameter Modes mode=Modes.Nominal;

   // algorithms and equations of _famefaults_c
   equation
    if mode==Modes.Fatigue then
     y = u*(1-amount);
    else
     y = u;
    end if;
   end _famefaults_c;

  // components of Spring
   Modelica.Mechanics.Rotational.Interfaces.Flange_a flange_a
           "Left flange of compliant 1-dim. rotational component";
   FAME.Dampers.RotationalWithoutConnectEquationsAasP _famefault_flange_a(amount=0.0);
   Modelica.Mechanics.Rotational.Interfaces.Flange_b flange_b
           "Right flange of compliant 1-dim. rotational component";
   FAME.Dampers.RotationalWithoutConnectEquationsAasP _famefault_flange_b(amount=0.0);
   parameter Modelica.SIunits.RotationalSpringConstant c(final min=0,start=1.0e5);
   _famefaults_c _famefault_c(u=c);
   Modelica.SIunits.Angle phi_rel(start=0)
                          "Relative rotation angle (= flange_b.phi - flange_a.phi)";
   Modelica.SIunits.Torque tau "Torque between flanges (= flange_b.tau)";
   parameter Modelica.SIunits.Angle phi_rel0=0 "Unstretched spring angle";

  // algorithms and equations of Spring
  equation
   tau = _famefault_c.y*(phi_rel-phi_rel0);
   phi_rel = _famefault_flange_b.port_b.phi-_famefault_flange_a.port_b.phi;
   _famefault_flange_b.port_b.tau = tau;
   _famefault_flange_a.port_b.tau = -tau;
   connect(flange_a,_famefault_flange_a.port_a);
   connect(flange_b,_famefault_flange_b.port_a);
  end Spring;
```

Figure 5: Fault augmented model for rotational spring.

DOI 10.3384/ECP14096437

[3] Wen Chen, M. Saif, and B. Shafai. Fault diagnosis in a class of differential-algebraic systems. In *American Control Conference, 2004. Proceedings of the 2004*, volume 5, pages 4398–4402 vol.5, 2004.

[4] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987. Also in: *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufmann, 1987), 280–297.

[5] Johan de Kleer, Bill Janssen, Daniel G. Bobrow, Tolga Kurtoglu, Bhaskar Saha, Nicholas R. Moore, and Saravan Sutharshana. Fault augmented modelica models. In *24th International Workshop on Principles of Diagnosis*, pages 71–78, Jerusalem, Israel, 2013.

[6] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, Piscataway, NJ, 2004.

[7] Mattias Krysander and Mattias Nyberg. xstructural analysis for fault diagnosis of dae systems utilizing graph theory and mss sets. Technical Report LiTH-ISY-R-2410, Department of Electrical Engineering, Linkoping University, 2002.

[8] Zsolt Lattmann, Adrian Pop, Johan de Kleer, Peter Fritzson, Bill Janssen, Sandeep Neema, Ted Bapty, Xenofon Koutsoukos, Matthew Klenk, Daniel Bobrow, Bhaskar Saha, and Tolga Kurtoglu. Verification and design exploration through meta tool integration with openmodelica. In *Proceedings of the 10th International Modelica Conference*, Lunde, Sweden, 2014.

[9] J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman. Population growth of human y chromosomes: a study of y chromosome microsatellites. *Molecular Biology and Evolution*, 16:1791–1798, 1991.

[10] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[11] D. Wegmann, C. Leuenberger, S. Neuenschwander, and L. Excoffier. Abctoolbox: a versatile for approximate bayesian algorithms. *BMC Bioinformatics*, 2(11):116, 1990.

[12] W. Zhang and D. J. Balding. Approximate bayesian computation in population genetics. *Genetics*, 27:2025–2035, 2002.