Enhancing the Simulation of Complex Mechanical Systems with Machine Learning

Victoria S. Harris

ATA Engineering, Inc.

***Abstract**: Submarine hatches are complicated mechanical systems with many moving parts and interfaces. As a mission-critical system, they currently require shock qualification with explosive testing: the hatch must not be damaged beyond the point of operability. This approach is expensive, complex, and can only address a small set of conditions. Abaqus is used to predict posttest operational effectiveness in lieu of test results, but the complexity of the system makes the process too time-consuming to positively affect the design process. As such, ATA Engineering is developing a methodology to reduce solve times for these complex systems and hence enable design guidance earlier in the design cycle.*

The analysis approach being developed uses fast-running metamodels to replace components of the system-level model that typically cause slow solution progress (e.g., contact regions). Detailed regions of the system-level finite element model are analyzed first in isolated breakout models, and machine learning algorithms are used to generate representative metamodels of those regions using information from the breakout analysis. These metamodels are then integrated into the system-level Abaqus model via user subroutines. The use of metamodels reduces the number of nonlinear interactions at the system level. This means that more analyses can be run and more damage conditions can be evaluated, enabling design changes to be introduced to the simulation quickly. The process provides the analyst with more information, more quickly, which will improve the overall design of the submarine hatch.

***Keywords:** Machine Learning, Operational Effectiveness, Damage, Submarines, User Elements*

1. Introduction

ATA Engineering (ATA) is developing a method for replacing sections of a complex kinematic model with a fast-running metamodel by way of user elements. The goal is to improve post-shock kinematic analysis of submarine hatches, but, when fully developed, the workflow should be applicable to a wide field of analysis problems.

1.1. Problem Background

The US Navy has rigorous requirements for surviving shock events for its critical systems on submarines (and other vessels). The systems must survive severe shock events and continue to operate effectively afterward to maintain mission capabilities. For submarine hatches, this means that the hatch must still open after sustaining damage. The Navy verifies operational integrity with shock testing followed by operational tests; however, these tests are expensive, time-consuming, and can only cover a limited set of shock environments.

Shipbuilders are currently using Abaqus for analysis to support the system certification: the combination of large articulated motions, contact, and friction make Abaqus the most effective tool currently available. Representing a system as complex as a submarine hatch, however, requires a detailed model using millions of nodes. This model is time-consuming to both build and run: a single simulation of post-shock operation can take a week to complete. Due to the modeling and computational expense, analysts can only consider a few different design configurations or damage conditions; the ability of analysis to drive design is limited.

1.2. ATA's Solution

ATA has been working on a method for more quickly getting information about the post-damage behavior of the submarine hatch by using machine learning algorithms to generate metamodels of system components. These metamodels represent the nonlinear behavior of the component in the system model, removing much of the computational expense due to friction and contact, with the goal of reducing overall solution time.

The general workflow of this procedure is as follows (explained in later sections):

1. Create a breakout model of the component to be replaced.
2. Run simulations on the breakout model with variations in loading and other parameters that are representative of the conditions in the system.
3. Use the data from those simulations to generate a metamodel of the component's behavior using machine learning.
4. Decide how to incorporate the user element into the system model.
5. Develop an algorithm that will correctly predict the behavior of the user element at each time step by feeding information from the system model into the metamodel, which generates new user element properties to be fed back into the system model.

This paper describes ongoing work; this workflow is the first step in developing a relationship between finite element analysis (FEA) and machine learning.

2. Machine Learning Background

Machine learning is a tool that allows predictions about future behavior to be drawn from existing data sets. It is used in everything from spam filters to self-driving cars. While the field has been around for decades, it has flourished over the last several years as computing power has increased and user-friendly toolkits have been developed in a variety of programming languages. We do not go into detail on machine learning theory and practice in this paper, but we do introduce the concepts necessary for understanding our implementation of it.

Machine learning can be broken down into three main steps: framing the problem, training an algorithm, and evaluating the resulting metamodel. The first step—framing the problem—may be the most important. Before you train a machine learning algorithm, you must determine what kind of question you are asking, what kind of data you need to answer it, and how you are going to collect that data.

In this project, we are looking to create a user element that accurately represents the behavior of a subsystem. For structural analysis, that user element may be a stiffness matrix, so we need to determine what affects the stiffness of the subsystem. Does it vary with loading? Over time? On contact properties? These factors are known as features. An algorithm can only predict the effects of the features it is given, so if an important feature is not included in the data set, the algorithm will not make good predictions. On the other hand, the more features included, the more data samples will be needed, which means longer run times. There are methods available for identifying the most important features in a data set, but these are outside the scope of this paper.

Once the features of interest have been identified, a data set must be generated for training the algorithm. In this project, breakout models are used to generate data about components. The breakout model typically has orders of magnitude fewer nodes and elements than the system model and runs in minutes rather than days. A set of feature values of interest is generated (e.g., friction coefficients between 0.05 and 0.5, bolt preloads between 0 ft-lb and 40 ft-lb), and the breakout model is analyzed with each combination of features. The results of those analyses become the data set for training the metamodel. An appropriate number of data points must be gathered to prevent overfitting of the data and accurately represent the input space (again, methods exist for testing the suitability of the size of the data set but are outside the scope of this paper).

Once the problem has been framed, the next step is to create the metamodel using a machine learning algorithm. There are many algorithms to choose from, ranging from fairly simple linear regression to the multilayered neural nets of deep learning. Algorithms can also be combined in a variety of ways to produce models that use the best parts of multiple algorithms. Most popular algorithms are available in preprogrammed toolkits like scikit-learn; one only needs to provide an appropriately formatted data set and select values for the algorithm's parameters. For this paper, four different algorithms were tested on the data, and each algorithm's parameters were tuned to produce optimal results.

The final step for any machine learning metamodel is verification. In the most general terms, before training an algorithm, the data set is split into "training" and "test" sets. The metamodel is trained using only the training data and subsequently used with the test data inputs to make predictions that are compared to the test data outputs. The metamodel's performance is evaluated based on its prediction accuracy (there can be more to this process, but it is again out of the scope of this paper). For this project, the metamodel can also be tested indirectly: the Abaqus solution

using the metamodel-powered user element can be compared to the results from the original Abaqus model. A bad metamodel will result in a poor comparison.

There is much more to the field of machine learning than this basic framework of framing a problem, selecting and training an algorithm, and verifying the results, but these are the basic steps followed in this paper. Starting in section 4, we describe our implementation of this workflow on an Abaqus assembly model representing a submarine hatch.

3. ATA's Notional Submarine Hatch Model

To put the proposed workflow into practice, we created a notional mechanical system that represents a simple submarine hatch (Figure 1). The system consists of a base on which the system is mounted, a lid, and a linear actuator/linkage mechanism to open and close the lid. While this system is much simpler than an operational submarine hatch, it includes a two-linkage mechanism that is a common feature of such hatch systems.

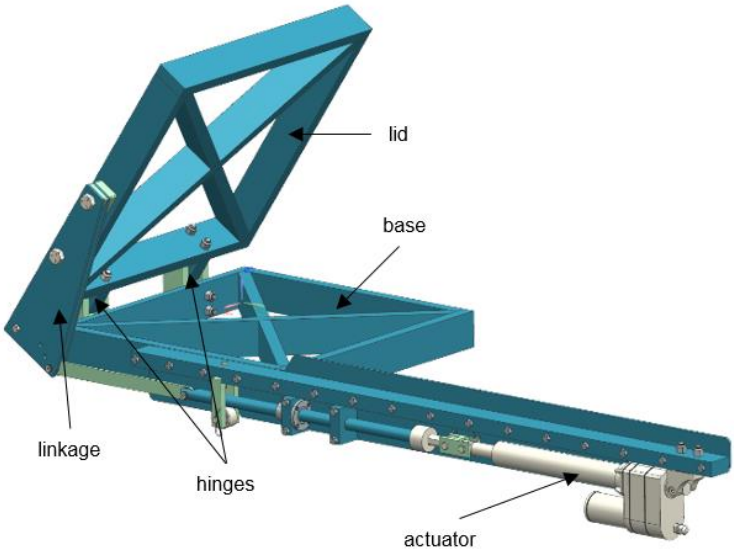

Figure 1. Design of notional submarine hatch.

The design includes several different configurations that represent "damage conditions"—slight variations in the design that are intended to mimic the damage inflicted on a submarine hatch due to a shock event, typically making the hatch harder to open. These variations mostly arise from different configurations of the hinge, which is shown in Figure 2. The hinge is designed with a pivot mechanism that acts as both a pin and a bolt; preload can be applied to the pin to clamp the lower hinge block's clevis ears tighter to the upper block. The amount of preload applied to the pin dictates the amount of friction force in the joint. Applying a high preload is representative of some kind of damage or wear in the hinge. The mounting points on the hatch lid and base are also designed with enough clearance that the hinge can be mounted out of alignment with the hatch frame. This causes binding in the hinge that again mimics the effects of post-shock damage.

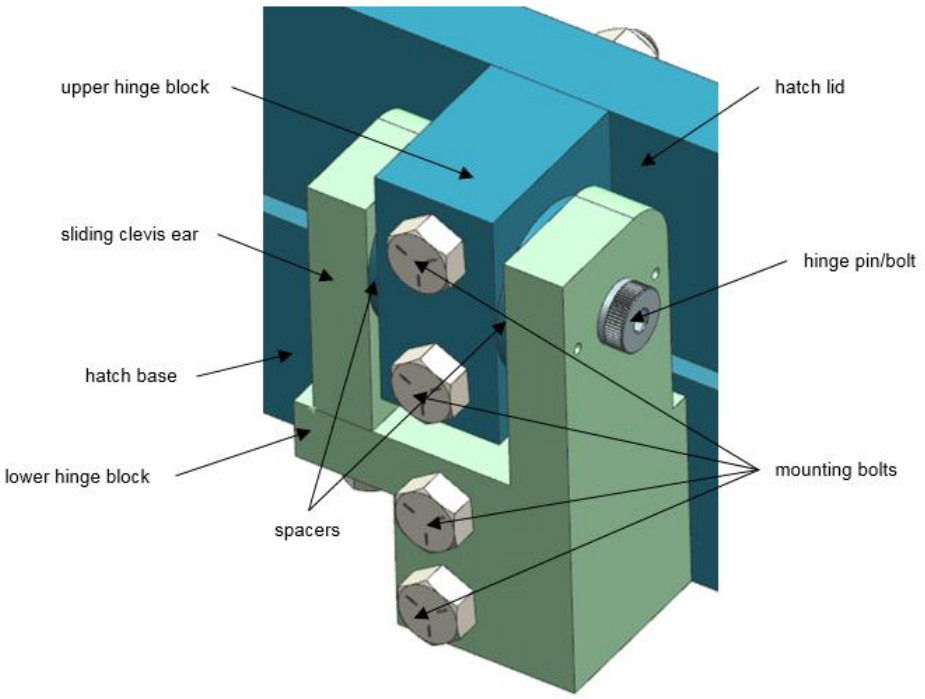

Figure 2. Hinge design.

An Abaqus finite element model (FEM), shown in Figure 3, was constructed for this design. The primary intent of the model is to extract kinematic behavior and linkage forces, so many regions (like the hatch base) were coarsely modeled to reduce analysis time. The hinges were of greatest interest for our analysis, so they were modeled in greater detail. Friction coefficients were taken from test results. The actuator was modeled as an Abaqus connector so that relative motions could be enforced. Multiple versions of this model corresponding to the different damage configurations described above were analyzed; they are referred to as the baseline models to which all results from the machine learning efforts will be compared.

The goal of the Navy's shock testing is to determine whether the submarine hatch will still open after taking damage, and the purpose of these FEMs is the same. The desired output from the analysis therefore is the actuator force required to open the hatch, which may or may not exceed the actuator hardware limits. The Abaqus analysis solution consists of a series of static steps to preload the hinge pins, and subsequent static steps to open and then close the lid with enforced connector velocity on the actuator connector.

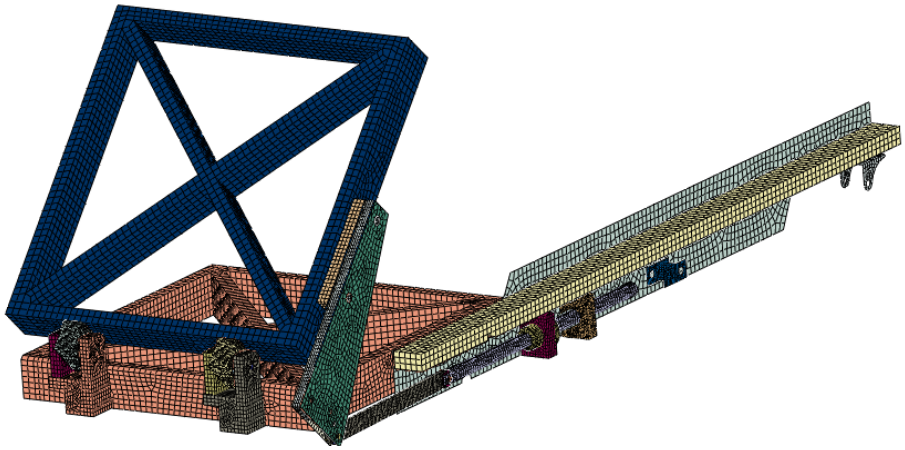

Figure 3. Abaqus FEM of notional submarine hatch test article.

4. ATA's FEM–Metamodel Workflow

As previously described, the aim of this work is to introduce machine learning into FEA to produce a new analysis method that will reduce the solution time needed for complex mechanical systems. The notional hatch FEM described in section 3 is not particularly complex but is being used to demonstrate proof of concept. The following sections describe how the individual steps of the workflow outlined in section 1.2 are used to replace the explicitly modeled hinges in that FEM with machine-learning-generated metamodels.

4.1. Develop the Breakout Model

In the notional hatch FEM, the most complex region is at the hinges. The mesh is densest there to properly account for contact between the pin and hinge blocks, with five pairs of contact surfaces defined on each hinge. Because of this, and because there are two identical hinges that can be represented with the same breakout model, the hinge was the component chosen to be replaced by a metamodel. The rest of the hatch was deliberately modeled with no contact, with the exception of initial contact between the hatch lid and base, so that when the hinges are replaced, the amount of contact remaining in the model is minimal. The reduction in the number of nodes, contact elements, and total equations can be seen in Table 1. The reduction in total equations is only 22.6%, which is not a large model reduction but enough to demonstrate the concept.

Table 1. Reduction in nodes, internal contact elements, and number of equations due to use of user elements at hinges.

	Baseline Model	UEL Model	% Reduction
Number of Nodes	199,441	101,373	49.2
Number of Internal Elements Generated for Contact	43,728	6,418	85.3
Number of Equations	333,594	258,238	22.6

The breakout model used to create the metamodel consists of the upper and lower hinge block, the pin, and the spacers from the baseline FEM, as shown in Figure 4. Before we could analyze the breakout model, we had to decide what information we needed to create the metamodel via machine learning (i.e., we needed to frame the problem). The easiest way to implement a user element is with a stiffness matrix, so stiffness information about the hinge was the output required for the breakout model for this problem (for other problems, it may be appropriate to use something other than the stiffness matrix approach).

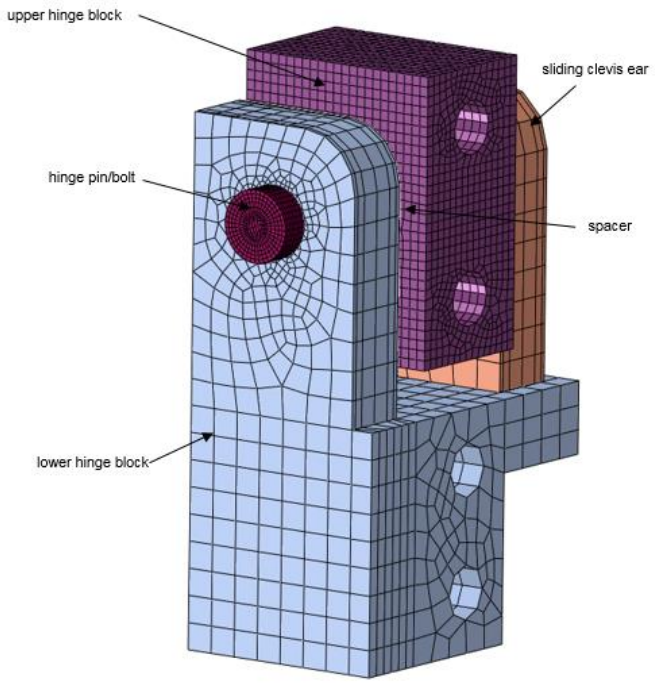

Figure 4. Hinge breakout FEM.

The next step was to determine what features of the hinge affect its stiffness. These features fell into two categories: features that vary during a single solution and features that stay constant for a single solution but vary across different hatch designs and/or damage conditions. The latter

category includes friction coefficients and hinge bolt preload, both of which affect the stiffness of the joint and must be properly accounted for in the breakout model but do not change during the opening of the hatch and so do not affect the stiffness function directly.

Investigation showed that the main feature affecting the stiffness of the hinge that varies during a solution is the moment about the pin axis applied to the upper hinge block. The data set used to generate a metamodel therefore had one input feature—applied moment—and one output feature—stiffness. Applied forces and moments in other directions also affect the stiffness but at a much smaller scale (at least for the damage cases related to varying hinge preload), so they were not included for the first pass at developing a metamodel. They will be added as features during future testing.

4.2. Generate a Data Set

To gather data for the machine learning algorithm from the breakout model, a moment is applied in a static step to the upper hinge block, and the resulting rotational displacement is recorded. Stiffness is then calculated by dividing the moment by the displacement. The moment is applied as a ramp, so a single solution produces multiple data points, each of which produces a new moment-displacement pair and corresponding moment-stiffness pair. For this analysis, with a single input feature and a relatively easily predicted function, around forty data points were collected (a more complex metamodel would require more data points).

The static solution fails when the upper hinge block starts to slip; dynamic analysis was used to verify that the applied moment at this point is in fact the slip moment. The post-slip stiffness is estimated from the slip moment and the velocity of the block rotation (this post-slip stiffness represents the small amount of resistance due to dynamic friction present in the moving joint). An example data set generated in this manner is shown in Figure 5.

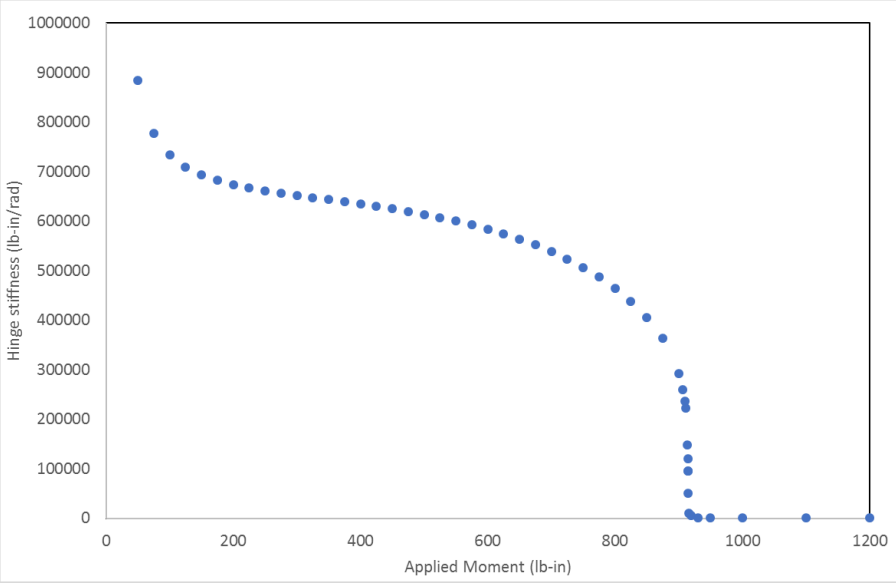

Figure 5. Applied moment vs. hinge stiffness (40 ft-lb preload).

4.3. Generate a Metamodel

The gathered data can now be used to train an algorithm to produce a metamodel. Since we only have one input feature, the data points shown in Figure 5 are enough to represent the relationship between moment and stiffness. As an added advantage, the data are simple enough that we can make a plot of our metamodel's predictions and visually inspect its accuracy.

Since these data were fairly simple, we used a collection of machine learning algorithms from scikit-learn.org to produce metamodels that can predict the stiffness of the hinge given a moment value. A method called cross-validation was used to iterate across the parameters of these algorithms and select the best settings for training. The best algorithm for these data is a decision tree, which uses if-then-else decision rules to approximate relationships between data (*Scikit-learn User Manual*, 2017). The estimated relationship between moment and stiffness for the example data from section 4.2 is shown in Figure 6.

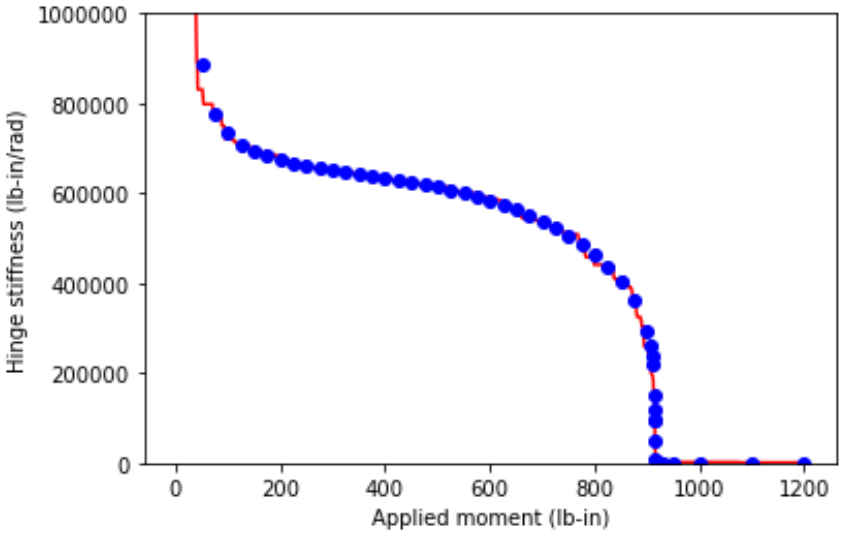

Figure 6. Moment-stiffness data set from breakout model (blue dots) and metamodel's predictions (red line).

4.4. Decide How to Incorporate the User Element into the System Model

In the next step, the placement of the user element in the system model must be chosen. A user element can have as many nodes as the user desires, but with more nodes comes more information to manage and a more complicated stiffness matrix to generate; it also drives up the solution time. To keep the notional hatch model as simple as possible, it was decided that the hinge pins and spacers would be removed from the FEM (as well as all of the contact surfaces between them). A point was placed at the center of the pin hole and connected to the upper hinge block's pin hole surfaces with a kinematic constraint, and a second, coincident point was connected to the lower hinge block's pin hole surfaces. Figure 7 shows an illustration of this setup. The user element definition between those two points follows. The setup for a single element is shown; the hatch

model used an element at each hinge, each with its own element type definition to allow independent stiffness updates.

```
*USER ELEMENT, TYPE=U98, NODES=          2, UNSYM
1,2,3,4,5,6
*ELEMENT, TYPE=U98
 99998, 21,20
*Elset, elset=user_elem
 99998
*UEL PROPERTY, elset = user_elem
```

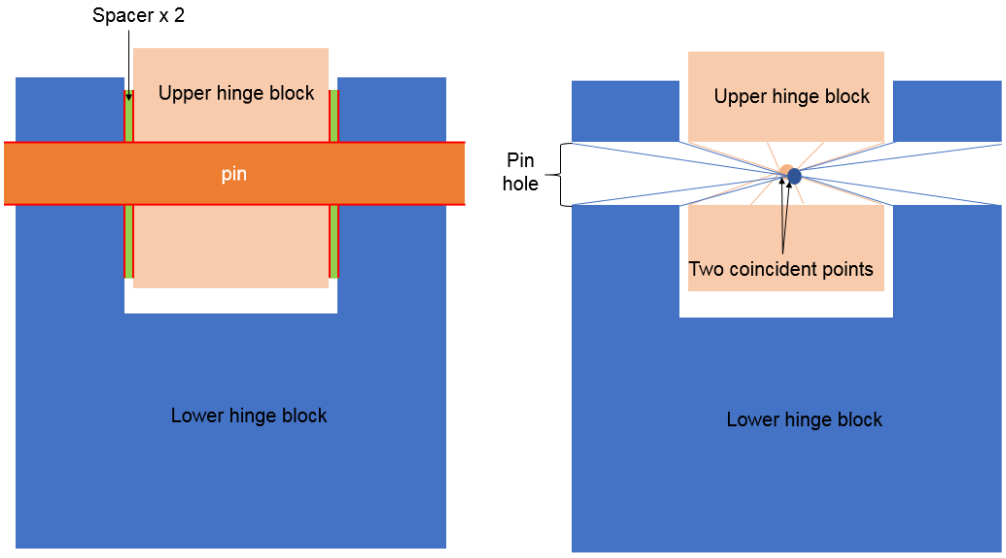

Figure 7. Hinge in baseline FEM (left) shows contact pairs as red lines; hinge in new model (right) places a user element between the two coincident points connected to the upper and lower hinge blocks.

4.5. Set Up the Workflow for the User Element

In the final step, the stiffness-predicting metamodel must be implemented via the Fortran subroutine for user elements, which is named UEL. For this workflow, we also use a second subroutine, named URDFIL, which makes results in the .fil file available to the user during the solve. A diagram of the workflow is shown in Figure 8.

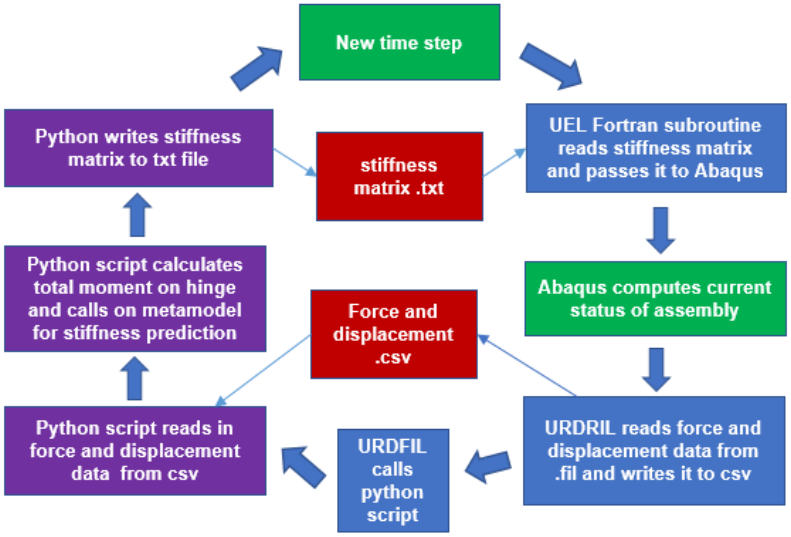

Figure 8. User element workflow for a single time step.

At the end of each time step, the URDFIL subroutine reads force and displacement data at the interface between the upper hinge block and the hatch lid and writes that data to a pair of CSV files. The subroutine then calls a Python script that reads in the force and displacement data and calculates the total moment on the hinge, and it then uses the metamodel to produce a corresponding rotational stiffness value. The script creates a 12 × 12 matrix to represent the overall stiffness of the hinge. The other diagonal terms (which are mostly independent of applied loads and preload/friction settings) were determined from separate static analysis of the hinge breakout model. This matrix is exported to a text file. At the beginning of the next time step, the UEL subroutine is called and reads in the stiffness matrix, which is inserted into the Abaqus model.

This routine originally led to undesirable oscillation of the stiffness value provided by the routine when the simulation encountered a state of hinge slip: after slip was reached, the stiffness would drop dramatically, which would cause the moment to drop, which made the stiffness rapidly increase, and the solution would fail. To smooth out the slip transition, stiffness increases were limited: now, the stiffness can no more than double in a single time step. This method damped the oscillation of the solution.

5. Results from the Notional Hatch Model

This workflow described in the previous section has been successfully implemented on the notional hatch model and used several times for different hinge bolt preload and friction settings. Two metrics for success were examined. First, since the purpose of submarine hatch analysis is to determine whether the hatch will still open in a damaged state, the maximum actuator operating force was recorded for each user element model and compared to an appropriate version of the

baseline model. Second, timing tests were done to compare the computational effort required for the metamodel-based user element models with baseline system models.

5.1. Actuator Force Comparisons

Table 2 shows the peak actuator forces for the baseline and user element models using several different hinge configurations. Two different coefficient of friction profiles were used that varied with normal pressure, labeled static and dynamic. Three different preloads on the hinge bolt were also used, ranging from 1600 to 6400 lb. The final column of the table shows the percent difference in the peak actuator force between the baseline version of the model and the user element version. Good correlation was found, ranging from at worst 13.6% to at best 4.3%.

Table 2. Peak actuator force comparisons for baseline models vs. UEL models.

Coefficient of Friction	Hinge Bolt Preload (lb)	Peak Actuator Force (lb)		
		Baseline Model	UEL Model	% Difference
dynamic	1600	525	503	−4.3
dynamic	4800	654	743	13.6
dynamic	6400	707	768	8.6
static	1600	558	509	−8.8
static	4800	728	792	8.8
static	6400	813	887	9.1

The actuator force through the entire hatch opening procedure was also compared. Some examples of actuator force functions are shown in Figure 9. As mentioned, the actuator force does oscillate in the user element model, and while it matches the peak force fairly well, the post-peak force tends to be underpredicted. We currently have plans for several possible improvements to the workflow and implementation details that are expected to improve correlation between the modeling procedures without sacrificing performance.

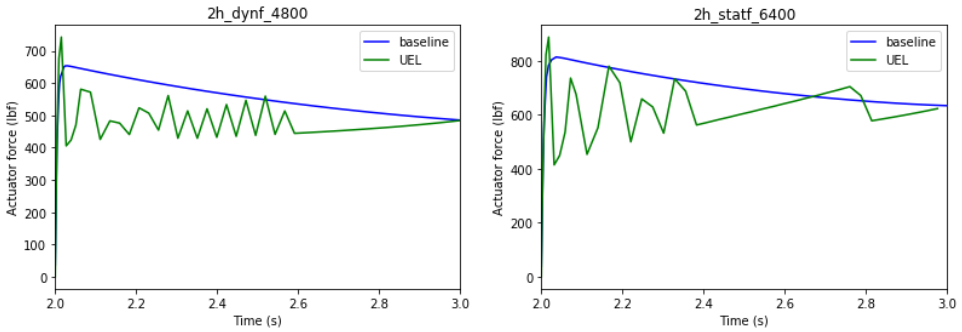

Figure 9. Actuator force vs. time for baseline (blue) and user element (green) models. Dynamic friction 4800 lb preload version shown left, static friction 6400 lb preload shown right.

5.2. Computational Effort Discussion

While the accuracy of the user element model is important, it is irrelevant if using the user element model does not improve upon the computational performance of the baseline model. To get a sense of the relative efficiency of the user element model, one user element model and its related baseline model were simulated multiple times in series on ATA's high-performance computing cluster. These simulations were performed at different times of the day, different days of the week, and with different cluster settings (including varying numbers of CPUs for parallel processing) in an attempt to clarify the effects of computational resource competition. The job time summaries were then compared among the solves.

The results showed that conditions on the cluster play a large role in the solution time for both the user element and baseline models. If the solves were running at the same time as other jobs, the solution times were highly variable, particularly for the user element model, making it difficult to draw strong conclusions. However, to isolate the performance measures, some jobs were run using all sixteen cores of a single computing node while the cluster was otherwise empty, and on these simulations, the user element model averaged 16,200 seconds of total CPU time whereas the baseline model used 25,300 seconds of total time—a difference of 36%. A second test using only four cores showed a similar solve time reduction of 41%. This test indicates that under perfect computing conditions, the user element method does reduce the solve time compared to the baseline model. At this time, more testing and development are needed to determine whether computational bottlenecks can be removed to further improve performance.

6. Conclusions and Future Work

ATA is developing a method for improving computational performance of complex kinematic models by replacing detailed sections of the model with machine-learning-powered user elements. The method has been demonstrated on a simplified notional actuated hatch model, and early results are promising. A set of peak actuator force comparisons between user element models and baseline models produced an average error of 8.9%, and early computational performance results show that the user element model takes less time to run.

There is plenty of room for improvement in these results. First, additional work will be put into improving the accuracy of the peak and post-peak actuator force results. Inclusion of more features in the stiffness prediction (e.g., forces and moments other than just the moment about the pin axis) should enable a more robust/accurate prediction.

This method also can and should be tested on a larger, more complex model. The notional hatch model has 200,000 nodes, but the analysis models built for the Navy's submarine hatches can have millions of nodes. Work has begun on constructing a more realistic hatch FEM to evaluate how the user element approach affects larger models.

Beyond these immediate changes, there are many possibilities for expanding on this work. The example in this paper is quite simple: hinge behavior is being represented with reasonable accuracy using a single two-dimensional curve. In reality, many more features affect the hinge's behavior—the other applied forces and moments discussed above, as well as things like the yield state of the joint, tolerance gaps, and so on—and there is potential for including much more information in the metamodel that represents the hinge.

Once a metamodel has been developed, it can produce information about a component much more quickly than FEA. Metamodels could potentially be used to derive statistical information about systems rather than just a single deterministic solution. For instance, it should be possible to quantify the effects of changing the hinge bolt preload on the behavior of the hatch system as a whole. On a larger model with multiple contributing metamodels, it may be possible to calculate the relative importance of damage to different components. Machine learning is a field with great potential, and its potential to assist FEA should be further explored.

7. References

1. *Scikit-learn User Guide*. Release 0.19.1. scikit-learn.org. 2017.

8. Acknowledgment

This material is based upon work supported by the Naval Sea Systems Command under Contract No. N00024-16-C-4006. ATA gratefully acknowledges the support of Mr. Randall Goodnight, Navy Technical Monitor.