Development of rapid error assessment tool for NASA multiscale analysis codes

Khoranhlai Anjuli Jones
Georgia Institute of Technology, Atlanta, GA, 303322
Xavier University of Louisiana, New Orleans, LA, 70125

Trenton M. Ricks
NASA Glenn Research Center, Cleveland, OH, 44135

Abstract

Modeling the effective behavior of advanced heterogeneous materials is a crucial step in their development. Thus, the micromechanics analysis code (MAC) based on the generalized method of cells (GMC) was developed. MAC/GMC is a design and analysis tool for metal matrix composites, polymer matrix composites, ceramic matrix composites, and smart composites and laminates (Bednarcyk, 2002). MAC/GMC was first developed over 20 years ago. Since then a multitude of features have been added, edited, or removed as needed with no way of validating results outside of classical debugging. Therefore, the need for a software verification and validation tool based on black box and white box testing became apparent. This paper focuses on a black box testing tool developed in the summer of 2018.

Keywords: software design; software verification; software validation; software testing; black box testing; multiscale analysis codes

I. Introduction

MAC/GMC is an analysis tool containing multiple micromechanics approaches. The approaches utilize each individual component's (subcell) material behavior. The macroscale response can then be determined by the geometric arrangement of the subcells and individual stress/strain response of the constituents. By utilizing MAC/GMC a user can analyze the response of a variety of material architectures to various thermal, mechanical, thermomechanical, and electrical load histories. The intent of the validation of MAC/GMC is not to confirm that it is valid over the complete domain of its intended applicability, but rather to prove that it is valid for a large set of known values. Although it is too costly and time-consuming to verify all possible outputs (Sargent, 2013), sufficient confidence can be gain by optimizing the amount of test coverage using equivalence partitioning. While there are many methods for software verification and validation, this paper focuses on black box testing for its superiority in analyzing the cohesion between functions in software.

Software testing is an essential part of software development, especially when said software is the basis of simulations models and executes heavy numerical computations. Testing techniques are primarily characterized by white box, black box, and grey box testing. White box testing evaluates the source code/internal design of the software typically via unit tests. Black box testing tests specific input and evaluates the results. Lastly, Grey box testing is a combination of both white box and black box testing.

There are several methodologies used to determine how code should be tested- either as a whole or by segments. Integrated testing tests the system as a whole. By contrast, unit testing tests small parts of the system creating fake input data (mocking) to ensure loose coupling. Tight coupling occurs when a group of classes (or methods) are heavily dependent on another. Loose coupling implies independence and is ideal when testing code. Loosely coupled classes enable the developers to isolate errors. The MAC/GMC Validation Tool uses both methodologies for different purposes. Integration is used to test MAC/GMC and unit testing is used to test the tool itself.

Lastly, the test cases in the MAC/GMC Validation Tool were generated using equivalence partitioning. Equivalence partitioning or equivalence class partitioning is a software testing designed to reduce the number of test cases evaluated while still maintaining sufficient software coverage (Juristo, 2012). Equivalence partitioning derives test cases from the input domain and partitions it by unit as equivalent data.

During the development of the MAC/GMC validation tool, every class and method was tested against variations of mock data based on a potential input. In addition, boundary conditions such as input size and numerical data boundaries were tested.

The MAC/GMC validation tool was developed in Java 8 using JetBrains IntelliJ IDEA. This integrated development environment (IDE) runs on Windows, macOS, and Linux and requires a minimum of 2 GB of RAM, 1.5 GB of hard disk space, and 1024x768 minimum screen resolution. However, this tool can function on any IDE that supports Java 8. This tool is currently open source and stored on GitHub under the name MACGMC_ValidationTool under the user anjulij. It can be cloned or downloaded by any user that wishes to validate their MAC/GMC output given that it falls within the range of examples stored in the project. In addition, beyond validation, the user can use the MAC/GMC validation tool to parse MAC/GMC into machine-readable data using a package within the project. This multiuse feature enables MAC/GMC users to utilize their data beyond the scope of the initial applications of MAC/GMC. Such applications include machine learning using different parsed output files as training sets.

II. Implementation

The MAC/GMC Validation Tool processes coupled output files and analyzes the differences between the two. Before analyzing the results of MAC/GMC for validation, its output must first be processed and parsed into a machine-readable format. MAC/GMC results are outputted as a text file segmented into sections based on the type of results being calculated. Typically, the first section echoes the input file, the second section displays the effective property results, and the third section displays the time-based output. The commonalities between sections are characterized by what one would recognize as equations, vectors, and matrices (table 1), therefore, additional section types can be constructed based on these types of data. In addition, some sections contained elements that are specific to MAC/GMC such as “tables” and “subcell failure” in *Effective Property Results* and *Time-Based Output* respectively (table 2). Each output file is stored in a multilayered map.

Equation	*x = 2 y = 7 z = 9 0 1*
Vector	*v: 1 2 3 w: 1 2 u: 1 8 0 1 3*
Matrix	*Matrix:* *1 2 3* *2 1 4* *2 1 4*

Table 1. *Contains generic sample data that can be found within MAC/GMC output*

Subcell failure	*[Several lines of Strings and Equations]*
Table	*(Label A, Label B) Label C Label D* *1 , 2 9.00 8* *2 , 4 8.02 7*

Table 2. *Contains sample data that can be found within MAC/GMC output that is specific to MAC/GMC*

Following parsing the output files, the MAC/GMC Validation Tool the coupled output maps are compared at each level and one of the following error messages is printed to a file for the user to read.

- **Error:** exception is raised in code being tested or the test itself. Essentially, the code did not execute properly.
- **Failure:** asserted condition is not true or expected exception is not raised. This will be raised for calculation errors.
- **Warning:** asserted condition is not true. This will be raised if a data type in the new output is not in the old output.
- **Pass:** the test did not fail or raise an exception. Either we need to make the testing more rigorous or pass the test suite.

When developing the MAC/GMC Validation Tool it was run through several benchmark tests and unit tests. Some of these tests focused on uniqueness because several values in a file may contain the same name. This could potentially result in an overwriting error due to how the data is stored. However, the multilayered aspect of the map has proven to be sufficient in overcoming this complication.

A. Parsing

A HashMap (figure 1.) is the primary data structure utilized in the MAC/GMC validation tool. A HashMap is an implementation of the Map interface based on hash tables. A Map is simply an object that maps keys to values, every key must be unique and can only map to one value. It is similar to a Dictionary in Python. A hash table operates by utilizing a hash function to compute hash values (hashes), which then function as indices of a set of objects (buckets). Each key and hash value serve as a unique identifier for a specific object that may be similar to other objects within the set. One can retrieve each value by iterating through the HashMap or by simply using the get operation if the key is known.

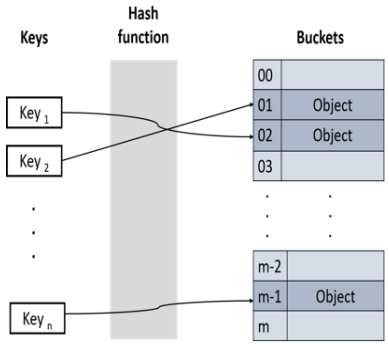

Figure 1. HashMap. *A hash function computes hashes, which function as indices to an array of "buckets" given a key and maps them to a value.*

In the MAC/GMC each output layer (figure 2.) is stored as a HashMap. Each layer is described in more detail below. The HashMap implementation is not only used to enable the user to analyze the data at any given layer but also uniquely identify each element in each file. When modifications are made to MAC/GMC they can affect the formatting this makes checking each file line by line inadequate. Unique identifiers

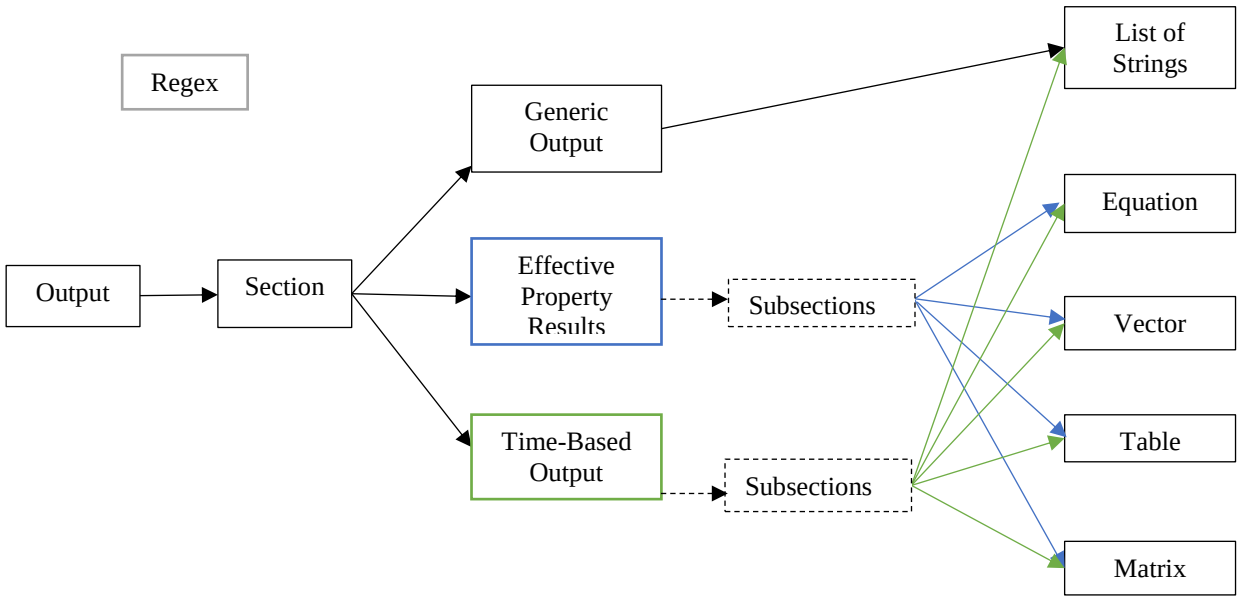

Figure 2. Parsing flow. *This figure represents the package "parsers" stored in the tool and the intended flow of execution. The output is a text file which is then parsed into a list of Strings, each line corresponding to an index in the list. Each layer functions independently as long as the input specifications are met i.e. a series of Strings to store as a list and an index to start and stop reading the list, or simply a single String in the case of an* `Equation` *or* `Vector`*. Each element is recognized by a regular expression, which can be modified in the static class Regex.java*

based on each values label creates a more accurate comparison between the two files. The MAC/GMC validation tool was also created with a focus on maintainability, therefore, each parser was generically designed within the constraints of efficiency. Each parser in the tool is stored within the package "parser". Every regular expression used in parsers is stored in the static class "Regex".

1. Output

```
Output(String filename)
```

`Output` is an object created from a MAC/GMC output file. It functions given a String which represents a file name. When initialized it iterates through the file and adds each line to a private list, `List<String> fileList`. `Output` also contains a private list of each section within the output file and a unique file name.

There are multiple methods that can be used to add each section to the section list. The first method iterates through the entire file list and isolates sections based on how sections are labeled (the regular expression is located in *Regex.java*). This method is not as efficient as other methods because it unnecessarily parses all of the data in a file. Alternatively, there are two methods which add sections given a specific section name. The first, `addSection` passes the name of the section to the `Section` class and adds the returned value to the list of sections. The second `addSectionEfficiently` passes the name of the section to an internal method to retrieve the section start and end index, passes that combination to the `Section` class, and then adds the returned value to the list of sections. `addSectionEfficiently` functions by skipping over sections that were previously added to the list of sections. Beyond efficiency `addSection` is beneficial in that `Section` elements are more loosely coupled to their corresponding `Output`. Output also contains two to String conversion methods, one that returns each `Section` name, start index, and index within the section list and another, which returns the parsed data as a String.

a. Section

```
Section(List<String> fileList, String sectionName)
Section(List<String> fileList, int sectionStartIndex, int sectionEndIndex)
```

A `Section` is an object of data from a MAC/GMC output file, unlike `Output` it only contains data from certain indices in the file. `Section` contains a private list of Strings that can represent the file that it is contained in, a String representing its name, and the indices on which it begins and ends. There are two constructors that can be used to implement `Section`. The first constructor only requires a list of string and a section name; it is ideal when treating a `Section` as an object separate from `Output`. However, when integrating each `Section` object with an `Output` object it is more efficient to use the constructor with parameters that include the start and end indices of the `Section` object.

When initialized the section boundaries are retrieved by iterating through each line in the file list until the line matches the regular expression for a section label, the line is then compared against the given section name and if the line contains the section name the start index is set as the line index. Similarly, the next line after the start index is the end index only if it matches either the regular expression for the end of the document or another section.

Subsequently, the data map of the section is retrieved based on the type of data it contains. If the section name matches the regular expression for effective property results, it retrieves its map of data based on the `EffectivePropertyResults` class; if the section name matches the regular expression for time-based output, it retrieves data based on the `TimeBasedOutput` class; if the object has not been specified it simply stores each line as a String. Each type of section is contained with "parsers.sections".

b. Effective Property Results

```
EffectivePropertyResults(List<String> fileList, int sectionStartIndex, int sectionEndIndex)
```

An `EffectivePropertyResults` is an object based on MAC/GMC output data from the *Effective Properties* section. An `EffectivePropertyResults` object consists of a file list, a start and end index, and a map representing the parsed data in that section. It contains three main subsections *Subcell Identification, Volume, and Material Arrangement* for each layer, *Effective Properties* at a particular temperature for each layer, and *Laminate Results* at a particular temperature.

Each `EffectivePropetyResults` object is initialized by iterating the file list until a line matches the regular expression of a generic label within the *Effective Property Results* section of a MAC/GMC output file, this label would indicate the beginning of a subsection. If the next line matches the regular expression signaling the end of the document, the loop of the section is exited. This class is complete with a to String converter to output the data map as a String and a method to retrieve the data map as a HashMap. It is complete with a to String converter, which returns the String value of the section data map.

i. Effective-Properties subpart

```
EffectivePropertiesSubpart(List<String> fileList, int subsectionStartIndex, int
sectionStartIndex, int sectionEndInex)
```

An `EffectivePropertiesSubpart` is an object that represents a subsection with the Effective Property Results section of MAC/GMC. It essentially a map where the key is the subsection label and the value is another map of each object within the subsection.

`EffectivePropertiesSubpart` is initialized by retrieving each object within the subsection. Each object within the subsection is retrieved by iterating through the file list until certain regular expression indicating that a certain object has been identified. After the object is retrieved the index is updated and the method continues through the loop until another subsection is found, the document end is reached, or the section end index is reached.

Effective properties subsections can contain a `Table` consisting of the subcell number, subcell material, and subcell volume, which is recognized by its label; `Equations`, which are recognized by an containing an equal sign followed by (a) number(s); `Vectors`, which are recognized by containing a colon followed by (a) number(s); and/or a `Matrix`, which is recognized as being a series of numbers. Each object is then retrieved and stored in the map corresponding to the subsection- details of the retrieval will be explained in a later section. It is complete with a to String converter, which returns the String value of the subsection map.

ii. Time-Based Output

```
TimeBasedOutput(List<String> fileList, int sectionStartIndex, int sectionEndIndex)
```

`TimeBasedOutput` is an object based on MAC/GMC output data from the *Time-Based Output* section. TimeBasedOutput consists of a file list, a section start and end index, and a map representing parsed data within the section. It contains one type of subsection, which is called *TimeStep* in the MAC/GMC validation tool.

A `TimeBasedOutput` object is initialized by retrieving each subsection within the section boundaries. To do this it iterates through the file list until a *TimeStep* is recognized by its corresponding regular expression. The subsequent method starts at the aforementioned index until a line matches the regular expression for one of the objects within the *Time-Based Output* section. Time-based output subsections can contain a `subcellFailure` consisting of a series of lines which may or may not contain equations, which is recognized by its label; a `Matrix`, which is recognized as being a series of numbers; `Equations` that are not in `subcellFailure`, which are recognized by an containing an equal sign followed by (a) number(s); and/or `Vectors` that are not in `subcellFailure`, which are recognized by containing a colon followed by (a) number(s). It is complete with a to String converter, which returns the String value of the section data map.

c. *Subparts*

The objects below are those that are completely generically parsed and are thus contained in "parsers.subparts".

i. Math objects

```
MathObect(String line)
```

A `MathObject` is a package-private interface used to parse any String that "looks like" a `Vector` or an `Equation`. This string can contain multiple objects of the same type and will still be parsed into a HashMap, mapping each equation or vector label to a numeric value (figure 3 and figure 4). It functions by splitting a trimmed String into a what is called a *line array* on spaces. Each value of the *line array* iterated through until the value at that particular index matches an equal sign or a colon. Every value before the equal sign or colon is labeled as the object name (key) and every subsequent numeric value is added to a list, this list of numbers is the value that assigned to the object name.

ii. Equations

```
Equation(String line)
```

An `Equation` extends `MathObject`. In addition, it has a unique to String converter, which essentially converts the Equation to its original form, a String. Before any of the methods are implemented the String is first checked by a method which returns true if it matches the regular expression for equations and false if it is anything else.

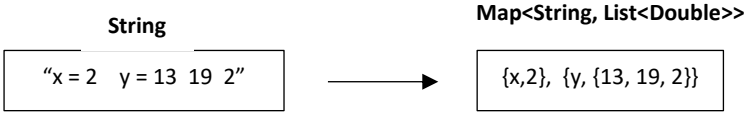

Figure 3. Equation examples

iii. Vectors

`Vectors(String line)`

A `Vectors` extends `MathObject`. In addition, it has a unique to String converter, which essentially converts the `Vector` to its original form, a String. Before any of the methods are implemented the String is first checked by a method which returns true if it matches the regular expression for vectors and false if it is anything else.

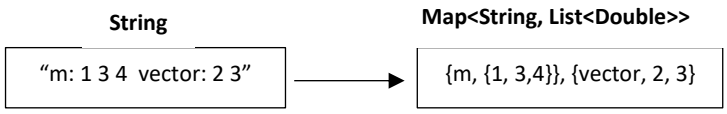

Figure 4. Vector examples

iv. Matrix object

There are several different methods that can be used to retrieve the indices which correspond to the beginning of a matrix. These two different methods are essential because the label for a matrix may be separated from the matrix values by several blank lines, because of this two different class were created, which are described below.

`MatrixStartIndicesRetrieval()`

`MatrixStartIndicesRetrieval` has a default constructor and its primary function is to sever as a retrieval method for the indices corresponding to the beginning of the matrix. It is package-private and every matrix is added to this static list of maps.

`MatrixStartIndex(List<String> fileList, int sectionStartIndex, int series_of_numbers_index)`

`MatrixStartIndex` is an object which consists of a file list from which the matrix data is read, the index of the beginning of the section in which the matrix is contained. This class retrieves the start indices when given the index of the line in the section containing a series of numbers. Given this index it iterates backward through the file until it encounters the first non-blank line or the beginning of the section, this line is the index of the label. This information is then stored in a map, the keys "Matrix Label Index" and "Matrix Values Start Index" corresponding to their respective integer values.

`SectionMatrixStartIndices(List<String> fileList, int sectionStartIndex, int sectionEndIndex)`

`SectionMatrixStartIndices` is an object that consists of a file list from which the matrix data is read. Additionally, it contains the index of the beginning and end of the section in which the matrix is contained. SectionMatrixStartIndices functions by iterating through a section treating each non-blank line as a potential matrix label. If the first non-blank line is a series of numbers, then that index is the beginning of a matrix; if it is not, then the process starts again until the entire section is analyzed.

`Matrix(List<String> filesList, int sectionEndIndex, Map<String, Integer> startIndices)`

`Matrix` is an object constructed from a file list, the index of the beginning and end of the section in which the matrix is contained, and a map of the start indices which correspond to the index that the label and be found and where the matrix values begin. A `Matrix` is characterized by a map corresponding to matrix information: the number of columns, the number of rows, the matrix title, and the 2D array in which the matrix values are stored. The primary method in the class is the one that retrieves the matrix information. The pseudocode for the matrix information retrieval is below.

```
matrixEnd ← 0
rows ← 0

columns ← the length of the element array at the matrix start index split on spaces

for i ← matrixStart; i < sectionEndIndex; i ← i+1
        String line ← the element at index i in the file list
        if the line at this index is not a series of numbers then
                matrixEnd ← i - 1
              rows ← (matrixEnd - matrixStart) + 1
              break
       end
end
```

After this information is retrieved a 2D array is created based on the number of columns and number of rows. It is then filled until the matrix is index corresponding to the end of the matrix is reached.

v. Table object

`Table(List<String> fileList, int tableStartIndex, int sectionEndIndex)`

`Table` is an object constructed from a file list, an index indicating where on the file the file begins, and an index indicating where the section ends. The start indices which correspond to the index that the label and be found and where the table values begin are found in a similar fashion to the method described in `MatrixStartIndex`. After these indices are found each subsequent line is considered an element of the table data until the next blank line. Each line is parsed either by storing a single number or multiple numbers that were separated by a comma.

vi. Subcell failure object

`SubcellFailure(List<String> fileList, int sectionEndIndex, int start)`

`SubcellFailure` is an object specific to MAC/GMC output. It is constructed by iterating through the file list and either storing the line as an `Equation` or a String. This loop ends on the next blank line.

B. Comparison

1. Output comparator

`OutputComparator(Output output_1, Output output_2)`

`OutputComparator` is an object which iterates through each layer of two output data maps and compares the values for errors (figure 5). These errors are then printed to a file.

a. Vector Comparator and Equation Comparator object

`VectorComparator(Vectors v1, Vectors v2)`

`EquationComparator(Equation e1, Equation e2)`

The `VectorComparator` and the `EquationComparator` function in a similar manner. They run through four checks: if they have the same labels (keys), have the same number of "Vectors" or "Equations" within the map, contain the same number of elements, and then if the elements are same value within a certain tolerance.

b. Matrix Comparator object

`MatrixComparator(Matrix m1, Matrix m2)`

The `MatrixComparator` functions by comparing the size of the matrices and the element value within a calculated tolerance given that the matrices have the same name.

c. Table Comparator object

`TableComparator(Table t1, Table t2)`

The `TableComparator` functions by comparing the size of the tables and the element value within a calculated tolerance given that the matrices have the same name. The first comparison is done between the number of lines in each table; the second comparison compares the number of elements on each line; and the last comparison is done between each numerical value in their corresponding tables.

d. Subcell failure Comparator object

`SubcellFailure(SubcellFailure s1, SubcellFailure s2)`

The `SubcellFailureComparator` functions by comparing each element in the lists corresponding to each `SubcellFailure`. They are first compared by object type, then by value.

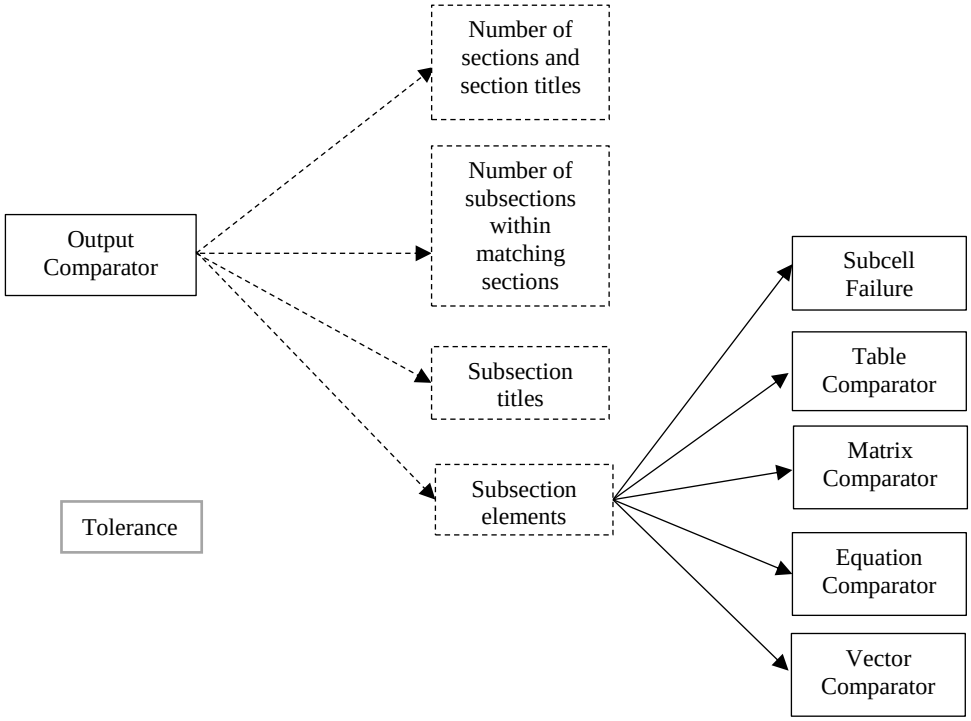

Figure 5. Comparison flow. *This diagram indicated the flow of comparison. The dashed boxes represent methods and the solid boxes represent classes.*

III. Further Studies and Discussion

Sample results can be found in the appendix below. In future, the MAC/GMC validation will be developed into a fully functional plugin. Ideally, the parsing package would store the data in an exportable machine-readable format such as JSON or a simple CSV.

Appendix

A. Sample output file

```
NASA Glenn Research Center
          Ohio Aerospace Institute

 Point of Contact:
   Dr. Steven M. Arnold
   NASA Glenn Research Center
   http://www.grc.nasa.gov/WWW/StructuresMaterials/MLP/software/mac-gmc/index.html

  ================================================================================================
                        *****              Section I:  Problem Input Data              *****
  ================================================================================================

 No Multiscale Modeling Options

================================================================================================
                        *****            Section II:  Effective Property Results           *****
================================================================================================

------------------- SUBCELL IDENTIFICATION, VOLUME, AND MATERIAL ARRANGEMENT FOR LAYER 1--------------------


    ALPHA     SUBCELL #     SUBCELL MATERIAL    SUBCELL VOLUME

      1        4             5          0.6500E+00
      2        3             4          5.1562E+00
      2        3             2          0.1562E+00
      2        4             2          0.3755E-01


                          TOTAL VOLUME =  0.1000E+01

      MATERIAL NO.  = 1    VOLUME RATIO = 0.6500E+00

      MATERIAL NO.  = 2    VOLUME RATIO = 0.3500E+00

      H = 0.1000E+01    L = 0.1000E+01

      Subcell Dimensions:
      h = 0.80623E+00 0.19377E+00
      l = 0.80623E+00 0.19377E+00

------------------- SUBCELL IDENTIFICATION, VOLUME, AND MATERIAL ARRANGEMENT FOR LAYER 2--------------------


  (BETA , GAMMA)    SUBCELL #     SUBCELL MATERIAL    SUBCELL VOLUME

     1 ,  2          3             4          5.6500E+00
     1 ,  2          2             2          0.1562E+00
     2 ,  1          3             2          0.1562E+00
     2 ,  2          4             2          0.3755E-01


                          TOTAL VOLUME = 0.1000E+01

      MATERIAL NO. = 1    VOLUME RATIO = 0.6500E+00

      MATERIAL NO. = 2    VOLUME RATIO = 0.3500E+00

      RUC Dimensions: H = 0.1000E+01    L = 0.1000E+01

      Subcell Dimensions:
      h = 0.80623E+00 0.19377E+00
      l = 0.80623E+00 0.19377E+00
```

---------- EFFECTIVE PROPERTIES AT TEMPERATURE = 21.00 FOR LAYER 1----------

CG - Effective/Macro Stiffness Matrix

2570	4499	0.4499	0.000000000000000E+00	0.000000000000000E+00	0.000000000000000E+00
4499	7827	0.3706	0.000000000000000E+00	0.000000000000000E+00	0.000000000000000E+00
4499	3706	0.7827	0.000000000000000E+00	0.000000000000000E+00	0.000000000000000E+00
0.0000	0.0000	0.0000	0.191593138598126E+10	0.000000000000000E+00	0.000000000000000E+00
0.0000	0.0000	0.0000	0.000000000000000E+00	0.416645647260888E+10	0.000000000000000E+00
0.0000	0.0000	0.0000	0.000000000000000E+00	0.000000000000000E+00	0.416645647260888E+10

Effective Engineering Moduli

E11S = 0.253540457926203E+12
N12S = 0.3901311339
N13S = 0.3901311339
E22S = 0.605030326120948E+10
N23S = 0.4681843609
E33S = 0.605030326120947E+10
G23S = 0.191593138598126E+10
G13S = 0.416645647260888E+10
G12S = 0.416645647260888E+10

Effective Tangent Thermal Expansion Coefficients

-0.4724E-06 0.2663E-04 0.2663E-04

Local Q Stiffness For Layer 1 Integration Pt. 1

2.545E+11	2.369E+09	0.000E+00
2.369E+09	6.072E+09	0.000E+00
0.000E+00	0.000E+00	4.166E+09

Global Q Stiffness For Layer 1 Integration Pt. 1

6.072E+09	2.369E+09	-9.495E-01
2.369E+09	2.545E+11	-5.000E+01
-9.495E-01	-5.000E+01	4.166E+09

Local Q Stiffness For Layer 1 Integration Pt. 2

2.545E+11	2.369E+09	0.000E+00
2.369E+09	6.072E+09	0.000E+00
0.000E+00	0.000E+00	4.166E+09

Global Q Stiffness For Layer 1 Integration Pt. 2

6.072E+09	2.369E+09	-9.495E-01
2.369E+09	2.545E+11	-5.000E+01
-9.495E-01	-5.000E+01	4.166E+09

```
 ================================================================================================
                    *****          Section III:  Time-Based Output          *****
 ================================================================================================


  1 TIME:  1.0000D-01    TEMP:  2.3000D+01    TSTEP:  1.0000D-01
------------------------------------------------------------------------------------------------
     STRESS:  0.0000D+00  1.1602D-01  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00
     STRAIN: -1.9167D-07  2.0000D-05 -9.5223D-06  0.0000D+00  0.0000D+00  0.0000D+00
  IN. STRAIN:  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00
  TH. STRAIN:  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00

  TANGENT CTE:  -4.2125D-07  2.9217D-05  2.9217D-05  0.0000D+00  0.0000D+00  0.0000D+00
   SECANT CTE:  -4.2125D-07  2.9217D-05  2.9217D-05  0.0000D+00  0.0000D+00  0.0000D+00
   NOTE: TREF =   23.000

  STIFFNESS:
 0.237668602E+06   0.4347655228E-04   0.434764012E+04  0.000000000E+00  0.000000000E+00  0.0000000000E+00
 0.434765528E+04   0.7580890191E+04   0.365103796E+04  0.000000000E+00  0.000000000E+00  0.0000000000E+00
 0.434765528E+04   0.3651037967E+04   0.758089019E+04  0.000000000E+00  0.000000000E+00  0.0000000000E+00
 0.800000000E+00   0.0000000000E+00   0.000000000E+00  0.184504960E+04  0.000000000E+00  0.0000000000E+00
 0.000000000E+00   0.0000000000E+00   0.000000000E+00  0.000000000E+00  0.376359682E+04  0.0000000000E+00
 0.000000000E+00   0.0000000000E+00   0.000000000E+00  0.000000000E+00  0.000000000E+00  0.3763596820E+04


  8 TIME:  2.0000D-01    TEMP:  2.3000D+01    TSTEP:  1.0000D-01
------------------------------------------------------------------------------------------------
     STRESS:  0.0000D+00  2.3204D-01  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00
     STRAIN: -3.8334D-07  4.0000D-05 -1.9045D-05  0.0000D+00  0.0000D+00  0.0000D+00
  IN. STRAIN:  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00
  TH. STRAIN:  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00  0.0000D+00

  TANGENT CTE:  -4.2125D-07  2.9217D-05  2.9217D-05  0.0000D+00  0.0000D+00  0.0000D+00
   SECANT CTE:  -4.2125D-07  2.9217D-05  2.9217D-05  0.0000D+00  0.0000D+00  0.0000D+00
   NOTE: TREF =   23.000

  STIFFNESS:
 0.2376686055E+06  0.4347655282E-04   0.434765528E+04  0.000000000E+00  0.00000000E+00  0.0000000000E+00
 0.4347655284E+04  0.7580890190E+04   0.365103796E+04  0.000000000E+00  0.00000000E+00  0.0000000000E+00
 0.4347655282E+04  0.3651037967E+04   0.758089019E+04  0.000000000E+00  0.00000000E+00  0.0000000000E+00
 0.0000000000E+00  0.00000000-0E+00   0.000000000E+00  0.184504960E+04  0.00000000E+00  0.0000000000E+00
 0.0000000000E+00  0.0000000000E+00  0.000000000E+00  0.000000000E+00  0.37635968E+04  0.0000000000E+00
 0.0000000000E+00  0.0000000000E+00  0.000000000E+00  0.000000000E+00  0.00000000E+00  0.3763596820E+04
```

B. Sample Parsed data

```
Section name: Section II:  Effective Property Results, Section start index: 16, Section end index: 230
Section name: Section III:  Time-Based Output, Section start index: 230, Section end index: 283

Section II:  Effective Property Results

SUBCELL IDENTIFICATION, VOLUME, AND MATERIAL ARRANGEMENT FOR LAYER 2
TOTAL VOLUME
TOTAL VOLUME = [1.0]

H
H = [1.0]

MATERIAL NO.
MATERIAL NO. = [2.0]

VOLUME RATIO
VOLUME RATIO = [0.35]

h
h = [0.80623, 0.19377]

L
L = [1.0]

l
l = [0.80623, 0.19377]

[(BETA , GAMMA), SUBCELL #, SUBCELL MATERIAL, SUBCELL VOLUME]
Label: (BETA , GAMMA), SUBCELL #, SUBCELL MATERIAL, SUBCELL VOLUME
Table:
        [[1.0, 2.0], [3.0], [4.0], [5.65]]
        [[1.0, 2.0], [2.0], [2.0], [0.1562]]
        [[2.0, 1.0], [3.0], [2.0], [0.1562]]
        [[2.0, 2.0], [4.0], [2.0], [0.03755]]

SUBCELL IDENTIFICATION, VOLUME, AND MATERIAL ARRANGEMENT FOR LAYER 1
TOTAL VOLUME
TOTAL VOLUME = [1.0]

H
H = [1.0]

MATERIAL NO.
MATERIAL NO. = [2.0]

VOLUME RATIO
VOLUME RATIO = [0.35]

h = [0.80623, 0.19377]

L
L = [1.0]

l
l = [0.80623, 0.19377]

[ALPHA, SUBCELL #, SUBCELL MATERIAL, SUBCELL VOLUME]
Label: ALPHA, SUBCELL #, SUBCELL MATERIAL, SUBCELL VOLUME
Table:
        [[1.0], [4.0], [5.0], [0.65]]
        [[2.0], [3.0], [4.0], [5.1562]]
        [[2.0], [3.0], [2.0], [0.1562]]
        [[2.0], [4.0], [2.0], [0.03755]]
```

```
Section III:  Time-Based Output

1 TIME:  1.0000D-01    TEMP:  2.3000D+01    TSTEP:  1.0000D-01
SECANT CTE
SECANT CTE: [-4.2125E-7, 2.9217E-5, 2.9217E-5, 0.0, 0.0, 0.0]

STIFFNESS

237668.605555532 4.34765528064012E-5 4347.65528064012 0.0 0.0 0.0
4347.65528064012 7580.8901902715 3651.03796747862 0.0 0.0 0.0
4347.65528064012 3651.03796747862 7580.8901902715 0.0 0.0 0.0
0.8 0.0 0.0 1845.04960945746 0.0 0.0
0.0 0.0 0.0 0.0 3763.59682066816 0.0
0.0 0.0 0.0 0.0 0.0 3763.59682066816


TH. STRAIN
TH. STRAIN: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

NOTE  TREF
NOTE TREF = [23.0]

STRESS
STRESS: [0.0, 0.11602, 0.0, 0.0, 0.0, 0.0]

IN. STRAIN
IN. STRAIN: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

STRAIN
STRAIN: [-1.9167E-7, 2.0E-5, -9.5223E-6, 0.0, 0.0, 0.0]

TANGENT CTE
TANGENT CTE: [-4.2125E-7, 2.9217E-5, 2.9217E-5, 0.0, 0.0, 0.0]

8 TIME:  2.0000D-01    TEMP:  2.3000D+01    TSTEP:  1.0000D-01
SECANT CTE
SECANT CTE: [-4.2125E-7, 2.9217E-5, 2.9217E-5, 0.0, 0.0, 0.0]

STIFFNESS

237668.605555532 4.34765528064012E-5 4347.65528064012 0.0 0.0 0.0
4347.65528064012 7580.8901902715 3651.03796747862 0.0 0.0 0.0
4347.65528064012 3651.03796747862 7580.8901902715 0.0 0.0 0.0
0.0 0.0 0.0 1845.04960945746 0.0 0.0
0.0 0.0 0.0 0.0 3763.59682066816 0.0
0.0 0.0 0.0 0.0 0.0 3763.59682066816

Time elapsed: 1.7080 s
```

C. Sample compared output

```
Comparison between files/sampleFile/dummydoc.out and newFiles/sampleFile/dummydoc.out

1. Number of sections and section names check
        true, section names and number of sections match in each output file

2. Number of subsections check
        false
        Section II:  Effective Property Results
        files/sampleFile/dummydoc.out
        subsection cardinality does not match in output files
        [SUBCELL IDENTIFICATION, VOLUME, AND MATERIAL ARRANGEMENT FOR LAYER 8]

        true
        Section III:  Time-Based Output
        both output files contain matching subsection cardinality

3. Subsection names check
        false
        Section III:  Time-Based Output
        subsection names do not match in both output files

4. Subsection elements check
        false
        Section II:  Effective Property Results
        Subsection name: LAMINATE RESULTS AT TEMPERATURE =    21.00
        matrices do not have the same values: 930300000000.000000 != 130300000000.000000

        false
        Section III:  Time-Based Output
        Subsection name: 1 TIME:  1.0000D-01     TEMP:  2.3000D+01     TSTEP:  1.0000D-01
        matrices do not have the same values: 0.000043 != 4347.655281
```

IV. Acknowledgements

The author K.A. Jones acknowledges NASA Glenn Research Center for providing this summer research opportunity and financial support. In addition, K.A. Jones thanks Calvin R. Robinson and Matthew J. Piekenbrok for their continuous guidance throughout the development of this tool.

V. References

Bednarcyk, B.A. et. al. “MAC/GMC 4.0 User's Manual—Keywords Manual”. NASA/TM—2002-212077/VOL2, 2002.

Juristo, N., Vegas, S., Solari, M., Abrahao, S., & Ramos, I. “Comparing the effectiveness of equivalence partitioning, branch testing and code reading by stepwise abstraction applied by subjects.” In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Converence on,* pp. 330-339. IEEE, 2012

Sargent, R.G. “Verification and validation of simulation models”, *Journal of Simulation*, 7:1, pp. 12-24, 2013