

SANDIA REPORT

SAND2016-1305

Unlimited Release

Printed February 2016

Metrics for the Complexity of Material Models

Stewart A. Silling

Prepared by

Sandia National Laboratories

Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



Metrics for the Complexity of Material Models

Stewart A. Silling
Multiscale Science Department
Sandia National Laboratories
P.O Box 5800
Albuquerque, NM 87185-1322

Abstract

Quantitative measures are proposed for characterizing the complexity of material models used in computational mechanics. The algorithms for evaluating these metrics operate on the mathematical equations in the model rather than a code implementation and are different from software complexity measures. The metrics do not rely on a physical understanding of the model, using instead only a formal statement of the equations. A new algorithm detects the dependencies, whether explicit or implicit, between all the variables. The resulting pattern of dependencies is expressed in a set of pathways, each of which represents a chain of dependence between the variables. These pathways provide the raw data used in the metrics, which correlate with the expected ease of understanding, coding, and applying the model. Usage of the `ComplexityMetrics` code is described, with examples.

Acknowledgments

This work was performed under the Advanced Simulation and Computing (ASC) program at Sandia National Laboratories. Helpful discussions with Drs. Edmundo Corona, Eliot Fang, Kevin Long, Benjamin Reedlunn, and William Scherzinger are gratefully acknowledged.

Contents

1	Introduction	7
2	Definition of the model.....	9
2.1	Model variables.....	9
2.2	Formal dependency matrix	10
2.3	Conditional statements	10
2.4	Functions	11
2.5	Internal state variables.....	11
2.6	Associativity, volume, and population.....	12
3	Units of complexity	14
4	Finding the dependencies between variables	15
4.1	Dependence algorithm	15
4.2	Depth of pathways	16
5	Complexity metrics.....	21
5.1	Sequentiality	21
5.2	Formal complexity	21
5.3	Full complexity	21
5.4	Indirectness.....	22
6	<code>ComplexityMetrics</code> code usage.....	24
6.1	Building the code	24
6.2	Running the code	24
6.3	Code input	24
6.4	Code output	26
7	Examples	27
7.1	Example 1	27
7.2	Example 2	29
7.3	Example 3	31
7.4	Example 4	33
7.5	Example 5: Mooney-Rivlin model	36
8	Conclusions	42

1 Introduction

Excessive complexity of a material model in computational mechanics can add to the costs of developing and maintaining the model throughout its lifecycle. It can also add to the costs of evaluating parameters and applying the model, as well as hindering the understanding of how it works. However, there is no widely accepted definition of what complexity is for a material model, let alone an accepted set of metrics for measuring it. As a result, most people view complexity subjectively: “I know it when I see it.” Persons who have a deep understanding of the scientific meaning of a model naturally tend to see its underlying simplicity. Yet others who may need to code, apply, or maintain the same model may view it as more complex. The purpose of the work described in this report is to propose objective, quantitative measures for complexity of a model that reduce the need for subjectivity in assessments of how complex a model is.

By the time a model under development is implemented in a code, it may be too late to significantly influence its complexity. Therefore, the metrics proposed here were designed to operate on the mathematical equations that comprise the model, not on software. As such, these metrics are fundamentally different from the many software complexity metrics that have been proposed [1]. For the same reason, evaluation of the metrics does not require number crunching: the model is never actually applied to any data; only a formal statement of the equations is used. This makes the present work different from the use of statistical techniques that apply a model many times to determine the sensitivity of outputs to inputs [2].

The metrics proposed here attempt to quantify three key aspects of complexity:

- *Sequentiality*. This measures whether the outputs can be solved for one-by-one, or whether they need to be solved for simultaneously. The need to solve sets of equations simultaneously, particularly if the equations are nonlinear, is one aspect of complexity.
- *Formal complexity*. This metric simply counts the total number of symbols that are used in the equations of the model, other than standard mathematical symbols and constants. Formal complexity correlates with our subjective first impression of the complexity of a model when reading its documentation for the first time. However, it is incomplete because it says nothing about the hidden interdependencies between variables; hence the need for the additional metrics proposed here.
- *Full complexity*. This metric augments the formal complexity by including a measure of indirect as well as direct dependencies between variables. It includes, for example, the possibility that input variable x affects output variable y because it is used in many intermediate quantities that determine y through convoluted mathematical interrelationships. The heart of this metric is a new algorithm that uncovers all of *pathways* from each input variable and material parameter to each output, representing a chain of intermediate dependencies. Full complexity attempts to measure the level of intellectual effort needed to fully understand how all the variables in the model interact

with each other.

- *Indirectness*. This measures the extent to which the dependencies between variables in the model are hidden from plain view in the formal statement of the model. A high value of indirectness means that the dependence between inputs and outputs can follow multiple pathways through different equations and intermediate variables. A high value therefore suggests that the model may be harder to understand and debug than if the dependencies were stated more explicitly.

The metrics proposed here use only a formal statement of the model in terms of symbols, without including any aspect of its scientific content. For scientists and engineers who attempt to comprehend a model by developing an intuitive understanding of the physical meaning of each variable, this requires changing gears. For example, persons who are fluent in the usage of index notation might question why the symbol u_i should be treated as different from u_j for purposes of the complexity metrics even though they mean the same thing, that is, the components of the vector u . However, relying completely formal mathematical statements helps ensure objectivity and uniformity of the complexity metrics.

In addition to defining the algorithms that act on a formal model to determine the metrics, this report also documents the `ComplexityMetrics` code that evaluates them for specific models. Several examples illustrate the result that a model with three inputs and three outputs can have widely different complexity metrics, depending on the specifics of the equations. To illustrate the application of the method to a real-world material model, the final example determines the metrics for the Mooney-Rivlin rubber elasticity model as documented in the LAME material model library [3].

2 Definition of the model

The methods described below for determining the complexity of a model use the number of occurrences of the various *symbols* used in the model, rather than its scientific content. In each equation in the model, we count up the number of times each symbol appears. These sums are recorded in the formal dependency matrix, to be defined below. This matrix contains all the information that is used in evaluating the complexity of the model.

2.1 Model variables

It is assumed that the model can be expressed in the form of E equations:

$$\begin{aligned} 0 &= e_1(y_1, \dots, y_{N_y}; x_1, \dots, x_{N_x}; p_1, \dots, p_{N_p}), \\ 0 &= e_2(y_1, \dots, y_{N_y}; x_1, \dots, x_{N_x}; p_1, \dots, p_{N_p}), \\ &\vdots \\ 0 &= e_E(y_1, \dots, y_{N_y}; x_1, \dots, x_{N_x}; p_1, \dots, p_{N_p}) \end{aligned}$$

where y_1, \dots are the outputs, x_1, \dots are the inputs, and p_1, \dots are the parameters. Intermediate quantities, that is, variables that are computed internally within the model but not saved after the each evaluation of the model, are treated as outputs. It is helpful to treat all the outputs, inputs, and parameters collectively as the $N = N_y + N_x + N_p$ variables v_i expressed as a row vector:

$$\langle v \rangle = \langle y_1 \dots y_{N_y} \ x_1 \dots x_{N_x} \ p_1 \dots p_{N_p} \rangle.$$

Thus the model may be written as

$$\begin{aligned} 0 &= e_1(v_1, \dots, v_N), \\ 0 &= e_2(v_1, \dots, v_N), \\ &\vdots \\ 0 &= e_E(v_1, \dots, v_N). \end{aligned}$$

It is assumed that $E \geq N_y$. In most properly formulated models, $E = N_y$, but the case of inequality is included here for generality. The case $E < N_y$ is excluded because there is little hope of determining the outputs from a smaller number of equations.

2.2 Formal dependency matrix

Form the $E \times N$ formal dependency matrix D defined by

$$D_{ij} = \text{number of times that the symbol } v_j \text{ appears in equation } e_i.$$

For example, if the model has $E = 1$ and

$$e(y, x, p) = 5y + x + x \sin px,$$

then

$$\langle v \rangle = \langle y \ x \ p \rangle, \quad D = \begin{bmatrix} 1 & 3 & 1 \end{bmatrix}.$$

D_{ij} depends on the way the model is written, not on the true meanings of the equations. If, in the above example, the model were rewritten as

$$e(y, x, p) = 5y + x/2 + x/2 + x \sin px$$

then we would have

$$D = \begin{bmatrix} 1 & 4 & 1 \end{bmatrix}.$$

Another example to emphasize that we are concerned only with symbols, not scientific content, is given by

$$e(y, x_i, p_i) = -y + \sum_{i=1}^{10} p_i x_i,$$

for which

$$D = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}.$$

In this example, it does not matter that there are really ten values of x and of p that are involved. For purposes of studying the complexity of the model, only the symbols x_i and p_i are included.

2.3 Conditional statements

A special case arises when a model contains cases or branches. For example, consider the model given by

$$y_1 = \begin{cases} x_1 & \text{if } x_3 \leq 6, \\ x_2 + \sqrt{x_1} & \text{if } 6 < x_3 \leq 8, \\ x_1 + x_2 + 1/x_3 & \text{otherwise.} \end{cases} \quad (1)$$

In this case we treat the expressions to the right of each “if” as outputs whose value is either 0 or 1. This creates new outputs that are included in the formal dependency matrix. The above model is rewritten as

$$\begin{aligned} y_1 &= x_1 y_2 + (x_2 + \sqrt{x_1}) y_3 + (x_1 + x_2 + 1/x_3)(1 - y_2 - y_3) \\ y_2 &= \text{eval}\{x_3 \leq 6\} \\ y_3 &= \text{eval}\{6 < x_3 \leq 8\} \end{aligned}$$

where the notation $\text{eval}\{\mathcal{E}\}$ signifies the value of the Boolean expression \mathcal{E} , either 0 or 1. The formal dependency matrix is then written

$$D = \begin{bmatrix} 1 & 2 & 2 & 3 & 2 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

where the vector of variables is

$$\langle v \rangle = \langle y_1 \ y_2 \ y_3 \ x_1 \ x_2 \ x_3 \rangle.$$

Each case in (1) adds one equation and one output, resulting in an increase in all of the complexity metrics.

2.4 Functions

Another special case occurs if the model contains references to functions. For purposes of constructing the formal dependency matrix, the symbol representing the function is ignored, as though it were a numerical constant. If the function is *not* considered to be part of the model for purposes of the complexity metrics, then no other action is necessary (for example if it a standard function like $\sin x$).

If, on the other hand, the function *is* considered to be part of the material model, its definition is included in the D matrix. The symbol representing the function is the treated as an output. Its dummy arguments are treated as inputs. For example, the model defined by

$$\begin{aligned} y &= 1/x + f(x) \\ f(q) &= q^5 + q \end{aligned}$$

has variables

$$\langle v \rangle = \langle y \ f \ x \ q \rangle$$

and formal dependency matrix

$$D = \begin{bmatrix} 1 & 1 & 2 & 0 \\ 0 & 1 & 0 & 3 \end{bmatrix}.$$

2.5 Internal state variables

Some models include quantities that are not part of the useful output but are computed internally and need to be saved for the next time the model is evaluated for the same element. For example, a model might evolve the entropy over time, even though this quantity does

not appear in the balance of linear momentum or the balance of energy. This type of variable, which is called an *internal state variable*, is treated as both an input variable and an output variable, representing the old and new values respectively. This is different from an intermediate value that is *not* saved between time steps, which is treated only as an output variable.

For example, consider the following model with one output y and one input x (exclusive of the internal state variable), and parameter Δt . The model has an internal state variable s . Let s_- denote the value of s for the element in previous time step:

$$\begin{aligned} y &= s^2 \\ s &= s_- + x\Delta t. \end{aligned}$$

The variables are

$$\langle v \rangle = \langle y \ s \ x \ s_- \ \Delta t \rangle$$

and the formal dependency matrix is

$$D = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

2.6 Associativity, volume, and population

Define the operator \mathcal{B} by

$$\mathcal{B}u = \begin{cases} 1, & \text{if } u > 0, \\ 0, & \text{otherwise.} \end{cases}$$

By the obvious extension to matrices, \mathcal{B} is applied to each element, thus

$$\mathcal{B} \begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Define the *associativity matrix* B of the model by

$$B = \mathcal{B}D.$$

The associativity matrix is not used in the complexity metrics defined below, but it is mentioned here because it has an interesting similarity to the analogous quantity used in graph theory. In fact, a material model can be represented as a graph with the variables as vertices and the submodels (to be defined below) as edges. However, these concepts are not needed in the present work.

Define the *volume* of a matrix A by

$$\mathcal{V}A = \sum_i \sum_j A_{ij}$$

where the sums are taken over all the rows and columns. The volume of a matrix is the sum of all the elements in the matrix. Define the *population* of a matrix A by

$$\mathcal{P}A = \sum_i \sum_j \mathcal{B}A_{ij}$$

where the sums are taken over all the rows and columns. The population of a matrix is the number of positive elements in the matrix.

3 Units of complexity

Some of the complexity measures discussed below are expressed in terms of *complexity units* (CU). One CU is defined to be the incremental increase in complexity of a model due to the inclusion of *one* additional input variable or parameter to *one* equation, provided that this change affects only *one* output parameter. For example, changing the model from

$$\begin{aligned}0 &= e_1(y_1, x_1, p_1), \\0 &= e_2(y_2, x_2, p_2)\end{aligned}$$

to

$$\begin{aligned}0 &= e_1(y_1, x_1, p_1, p_3), \\0 &= e_2(y_2, x_2, p_2)\end{aligned}$$

increases the complexity measures (to be defined below) by 1 CU.

4 Finding the dependencies between variables

This section describes an algorithm for determining the following:

- a solution strategy that detects a sequence of *submodels* (sets of equations that can be solved by themselves from the inputs, the parameters, and the results of previously solved submodels);
- the *pathways* (sequences of intermediate dependencies between variables).

4.1 Dependence algorithm

The algorithm described here determines all the submodels and pathways. Let \mathcal{C}_m^n denote the set of all combinations of the integers $1, 2, \dots, n$ taken m ways. For example, the elements of \mathcal{C}_2^4 are

$$\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}.$$

The algorithm below divides the *solution* of the complete model (that is, determining all the outputs from given values of the inputs and parameters) into steps. Each step solves a *submodel* within D . A submodel \mathcal{K} is a subset of the model comprised of K equations each of which depends on the same L outputs, $L \leq K$. As each submodel is identified, all of its equations are thereafter considered to be *solved*, and all the outputs in those equations are considered to be *solved-for*. The algorithm, in search of the easiest route to solving the entire set of equations, first tries to find submodels with one unsolved equation (and possibly additional previously solved equations). It then considers this unsolved equation thereafter to be solved, and all the outputs it contains to be solved-for. When the algorithm can't find any more submodels with one unsolved equation, it searches for those with two, then three, and so on.

As it searches for submodels, the algorithm preferentially searches for those with the fewest numbers of outputs. In general, a submodel includes some equations that have previously been solved and always contains one or more that have not. Also in general, the variables in a submodel contain some outputs that have been solved-for and some that have not. The number of equations that are solved in submodel number U is denoted G_U , $G_U \leq K$. This number is important in some of the complexity metrics defined below, because a model *all* of whose submodels have $G_U = 1$ has a great advantage in simplicity.

The significance of this strategy of identifying the fewest number of unsolved equations that can be solved in each step is that it avoids unfairly assigning high complexity to portions of the model that can be solved individually in a simple way. The outputs that are *solved-for* in a submodel will never have additional complexity attributed to them by later submodels, even if the outputs *appear* in those later submodels.

A *pathway* is a row vector $R = \langle r_1 \ r_2 \ r_3 \ \dots \ r_\delta \rangle$ that represents the chain of intermediate dependencies by which variable r_1 depends on r_δ , where δ is the *length* of the

pathway. The meaning of the pathway may be stated as follows: “Variable r_1 depends on variable r_2 which depends on variable $r_3 \dots$ which depends on variable r_δ .”

A separate row vector $\langle U_1 \ U_2 \ \dots \ U_{\delta-1} \ 0 \rangle$ contains the submodel numbers through which r_n depends on r_{n+1} . By convention, the starting variable in a pathway is numbered highest, that is, output r_1 depends through the chain of variables on r_δ .

A summary of the notation that appears in the dependency algorithm is as follows:

- \mathcal{K} is a submodel.
- K is the number of equations in a submodel.
- U is a submodel number.
- L is the number of outputs in a submodel.
- C is the set of outputs in a submodel.
- G_U is the number of equations that are solved by submodel number U .
- R_n is pathway n , a row vector containing a chain of dependencies.
- U_n is a row vector containing the submodels in pathway n .
- P is the total number of pathways.
- δ_n is the length of pathway number n (number of elements in R_n).
- T is the set of variables that appear in a submodel.

The details of the dependency algorithm are given in Algorithm 1. An example of a full set of pathways for a model with two inputs and two outputs is illustrated in Figure 1 in the form of a directed graph.

4.2 Depth of pathways

For each pathway $n = 1, 2, \dots, P$, define the *depth* d_n by

$$d_n = \sum_{1 \leq i \leq \delta_n} G_{U_N}$$

To interpret this definition, recall that G_U is the number of equations that are solved in submodel number U . d_n is therefore the total number of equations that need to be solved to get from the start of the chain to the end, along this pathway. In general, the same variables can depend on each other through multiple pathways, some shorter or longer than others (Figure 1).

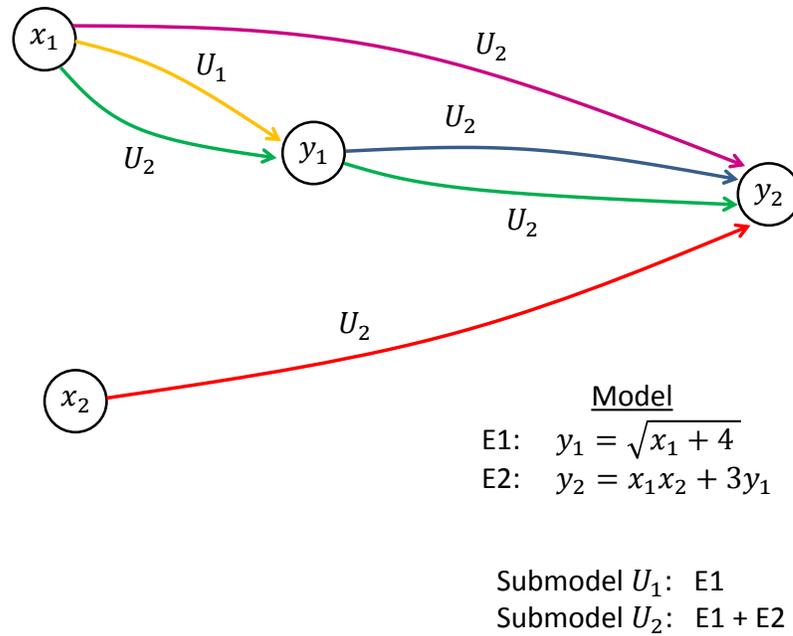


Figure 1. Full set of pathways in a model with two equations. Output y_2 depends both directly (purple pathway) and indirectly (green pathway) on input x_1 . It also depends on y_1 (blue pathway) although y_1 does not depend on y_2 . The green pathway has depth 2; all others have depth 1.

Algorithm 1 Find the dependencies of each output on all other variables.

```

1: procedure FINDDEPENDENCIES

2:    $\triangleright$  Initialize.
3:   All  $E$  equations are unsolved.
4:   All  $N_y$  outputs are unsolved-for.
5:    $U \leftarrow 0$ .  $\triangleright$  Submodel counter

6:    $\triangleright$  Define an initial pathway for each variable.
7:    $P \leftarrow N$   $\triangleright$  Pathway counter
8:   for each  $n \leftarrow 1, N$  do
9:      $R_n \leftarrow n$ .  $\triangleright$  Pathway variables
10:     $U_n \leftarrow 0$ .  $\triangleright$  Pathway submodels
11:     $\delta_n \leftarrow 1$ .  $\triangleright$  Pathway length
12:   end for

13:    $\triangleright$  Find all submodels, starting with those with the fewest
14:    $\triangleright$  unsolved equations.
15:   for each  $G \leftarrow 1, E$  do

16:      $\triangleright$  Find all submodels with  $G$  unsolved equations.
17:      $\triangleright L$  is the number of outputs in a submodel.
18:     for each  $L \leftarrow 1, N_y$  do

19:        $\triangleright$  Find sets of equations  $\mathcal{K}$  with the same  $L$  outputs
20:        $\triangleright$  whether solved-for or not.
21:       for each  $C \in \mathcal{C}_L^{N_y}$  do
22:          $\mathcal{K} \leftarrow$  set of equations whose outputs are all in  $C$ .
23:          $G' \leftarrow$  number of unsolved equations in  $\mathcal{K}$ .

24:          $\triangleright$  Only want submodels with  $G$  unsolved equations.
25:         if  $G' = G$  then
26:            $K \leftarrow$  number of equations in  $\mathcal{K}$ .

27:            $\triangleright$  Need at least as many equations as outputs.
28:           if  $K \geq L$  then  $\triangleright$  Solvable submodel found
29:              $U \leftarrow U + 1$ .  $\triangleright$  Submodel counter
30:              $\mathcal{K}_U \leftarrow \mathcal{K}$ .
31:              $G_U \leftarrow G$ .  $\triangleright$  Number of equations solved in  $\mathcal{K}$ 

```

Algorithm 2 Dependency algorithm, ctd.

```
32:         for each unsolved-for output  $j \in C$  do
33:              $\triangleright$  Create new pathways through all variables
34:              $\triangleright$  that  $j$  depends on.
35:             CREATEPATHWAYS( $j, \mathcal{K}$ )
36:             Output  $j$  is now solved for.
37:         end for
38:         for each unsolved equation in  $i \in \mathcal{K}$  do
39:             Equation  $i$  is now solved.
40:         end for
41:     end if
42: end if
43: end for
44: end for
45: end for
46: end procedure
```

Algorithm 3 Add pathways as each output becomes solved-for.

```
1: procedure CREATEPATHWAYS( $j, \mathcal{K}$ )
2:      $T \leftarrow$  set of all variables in  $\mathcal{K}$ .

3:      $\triangleright$  Create a new pathway to  $j$  from each existing pathway that ends
4:      $\triangleright$  in a variable (other than  $j$ ) in submodel  $\mathcal{K}$ .
5:     for each  $t \in T$  do
6:         for each  $n \leftarrow 1, P$  do
7:             if  $\mathcal{L}R_n = t$  then  $\triangleright$  Leftmost element of  $R_n$ .
8:                 if  $t \neq j$  then
9:                      $P \leftarrow P + 1$ 
10:                     $R_P \leftarrow j \parallel R_n$   $\triangleright$  New pathway is  $j$  appended by  $R_n$ .
11:                     $U_P \leftarrow U \parallel U_n$ .  $\triangleright$  Submodels in new pathway.
12:                     $\delta_P \leftarrow \delta_n + 1$ .  $\triangleright$  Length of new pathway.
13:                end if
14:            end if
15:        end for
16:    end for
17: end procedure
```

Higher values of depth tend to increase the complexity of the model, because they indicate that dependencies follow tortuous paths that involve interrelation of multiple equations in ways that may not be apparent to a person attempting to understand the model. Units of depth are CU.

5 Complexity metrics

This section describes the proposed complexity metrics, making use of the dependencies between variables determined by the algorithm described above.

5.1 Sequentiality

Recall that G_u is the number of equations that are solved in submodel u . Define the *sequentiality* of the model σ by

$$\sigma = \max_{1 \leq u \leq U} \{G_u\}$$

where U is the total number of submodels. If $\sigma = 1$, the model is *ideally sequential*, meaning that the outputs can be solved for one at a time. Otherwise, σ represents the largest number of equations that must be solved for simultaneously. Larger values of σ imply greater complexity because of the need to solve multiple equations simultaneously. Sequentiality is a dimensionless measure.

5.2 Formal complexity

Define the *formal complexity* ϕ_0 by

$$\phi_0 = \mathcal{V}D.$$

ϕ_0 is a measure of the total number of times all the variables appear in the formal statement of the model. It does not take into account the dependencies that are not stated explicitly in the formal statement. It represents the complexity of the model as it would apply to a person with a strong understanding of the physics represented by the model. To such a person, the indirect dependencies of the outputs on the other variables would not be essential to understanding how the model works. The units of formal complexity are CU.

5.3 Full complexity

Define the *full complexity* ϕ_1 by

$$\phi_1 = \phi_0 + \left(\sum_{1 \leq n \leq P} d_n + N_y - \mathcal{P}D \right). \quad (2)$$

ϕ_1 is a measure of the total number of dependencies of the outputs on the inputs and parameters, weighted by the depth of the pathways that characterize the individual dependencies. The full complexity can greatly exceed the formal complexity because, for a complex model, the number of pathways (and therefore the sum of their depths) can be much greater than the number of symbols in the formal statement of the model. The last two terms in (2),

$N_y - \mathcal{P}D$, appear as normalization terms such that $\phi_1 = \phi_0$ in the simplest possible models (see Example 1 below).

The full complexity is a measure of the total number of “pieces of information” (variables and the relationships between them) that are present, even if these relationships are not stated explicitly in the formal statement of the model. This metric therefore correlates with the “cognitive load” placed on a person, such as a professional programmer with no training in the physical sciences, who approaches the task of learning the model without the benefit of physical understanding. The units of full complexity are CU.

5.4 Indirectness

Define the *indirectness* of the model by

$$\omega = \frac{\phi_1}{\phi_0}.$$

This is a measure of the extent to which dependencies between variables are hidden, that is, not obvious from the formal model. A large value of ω indicates that changing the value of one input could result in changes in the outputs that are not obvious (except possibly to an expert in the physics) on the basis of the formal statement of the model. Indirectness is a dimensionless measure.

If $\omega = 1$, the model is *ideally direct*. This condition is met by model of the form

$$\begin{aligned} 0 &= e_1(y_1; x_1; p_1), \\ 0 &= e_2(y_2; x_2; p_2), \\ &\vdots \\ 0 &= e_{N_y}(y_{N_y}; x_{N_y}; p_{N_y}). \end{aligned}$$

A model can sometimes be restated in a way that reduces indirectness by “in-lining,” that is, writing out expressions explicitly to avoid the use of intermediate variables. For example, the model in Figure 1 can be rewritten as shown in Figure 2, resulting in fewer pathways. This reduces the indirectness from 1.33 to 1.00.

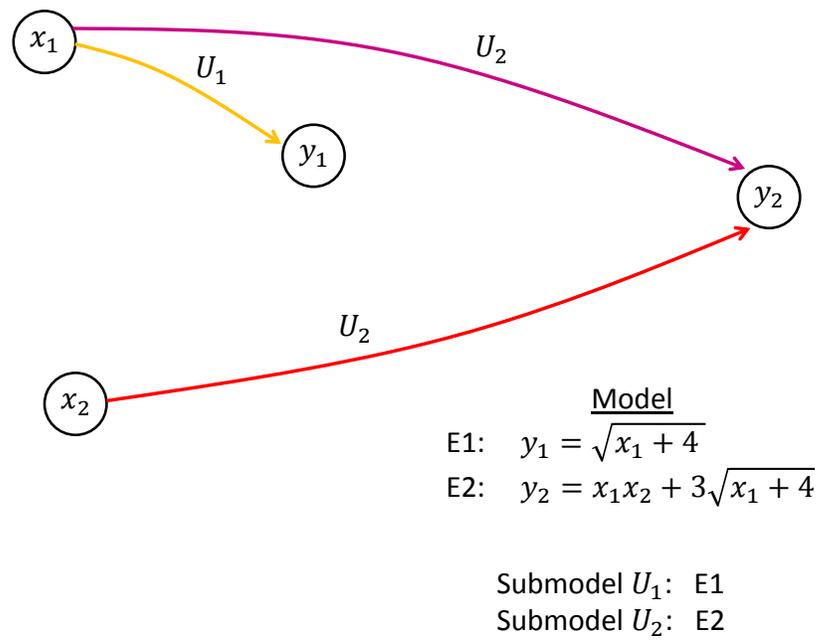


Figure 2. Rewriting the model in Figure 1 to avoid using y_1 as an intermediate variable reduces the number of pathways and the indirectness.

6 ComplexityMetrics code usage

The algorithms and complexity metrics described above have been implemented in the ComplexityMetrics code. The following is a summary of how to build and use the code.

6.1 Building the code

To build the code on a Linux system with an Intel Fortran 90 compiler type the following from a Linux command line:

```
ifort -o ComplexityMetrics ComplexityMetrics.f90
```

6.2 Running the code

To run the code, type the following from a Linux command line:

```
ComplexityMetrics [file.in]
```

where `file.in` is the text input file. The default input file name is `complex.in`.

6.3 Code input

The contents of the input file `file.in` are as follows. Keywords start in column 1. Numerical values can be entered in any format and start on the line below the keyword. All numerical values are non-negative integers.

```
number_of_outputs
```

```
Ny
```

where $Ny = N_y$.

```
number_of_inputs
```

```
Nx
```

where $Nx = N_x$.

number_of_parameters

N_p

where $N_p = N_p$.

number_of_equations

E

where $E = E$.

outputs

Y_1

Y_2

...

Y_{N_y}

where Y_1, \dots are N_y character strings identifying the outputs, one per line.

inputs

X_1

X_2

...

X_{N_x}

where X_1, \dots are N_x character strings identifying the inputs, one per line.

parameters

P_1

P_2

...

P_{N_p}

where P_1, \dots are N_p character strings identifying the inputs, one per line.

equations

ename1

ename2

...

enameE

where `ename1, ...` are E character strings identifying the equations, one per line.

```
formal_model
D11 D12 ... D1N
D21 D22 ... D2N
...
DE1 DE2 ... DEN
```

where D_{11}, \dots are the $E \times N$ elements of the formal dependency matrix D , where E is the number of equations in the model and $N = N_y + N_x + N_p$.

6.4 Code output

Code output is written to the standard output file. The output includes the following:

- A list of all the equations in each submodel, indicating which equations are solved by the submodel and which are not.
- A list of all the pathways to each output. For each element of the pathway, the list shows the number of the submodel that created it. The depth of each pathway is also given.
- The complexity metrics.

7 Examples

7.1 Example 1

Consider the following model with 3 inputs, 3 outputs, and 1 parameter:

$$\begin{aligned}0 &= Y_1 + 5X_1 + P_1 \\0 &= Y_2 + 2X_2 \\0 &= Y_3 - \sqrt{X_3}\end{aligned}$$

The variables are

$$\langle Y_1 \ Y_2 \ Y_3 \ X_1 \ X_2 \ X_3 \ P_1 \rangle$$

The code input is as follows:

```
number_of_outputs
  3
outputs
  Y1
  Y2
  Y3
number_of_inputs
  3
inputs
  X1
  X2
  X3
number_of_parameters
  1
parameters
  P1
number_of_equations
  3
equations
  E1
  E2
  E3
formal_model
  1 0 0   1 0 0   1
  0 1 0   0 1 0   0
  0 0 1   0 0 1   0
```

The code output is as follows:

List of the equations in each submodel:

Submodel 1:
E1 is solved by this submodel.
Submodel 2:
E2 is solved by this submodel.
Submodel 3:
E3 is solved by this submodel.

Paths to output 1 Y1
Y1 depends through submodel 1 on X1 (depth= 1)
Y1 depends through submodel 1 on P1 (depth= 1)
Paths to output 2 Y2
Y2 depends through submodel 2 on X2 (depth= 1)
Paths to output 3 Y3
Y3 depends through submodel 3 on X3 (depth= 1)

Net complexity measures:

Sequentiality	:	1
Formal model complexity	:	7
Full model complexity	:	7
Indirectness	:	1.00

The model in this example, in spite of having multiple inputs and outputs, is both ideally sequential and ideally direct. This indicates that there are no hidden dependencies, making it easy to debug any problems that may occur in applications. For example, if a finite element has a bad value of Y_3 , we don't have to look very far to find the culprit: it has to be X_3 .

7.2 Example 2

Compare the following model with Example 1:

$$0 = Y_1 + 5X_1/Y_3 + Y_3 + P_1$$

$$0 = Y_2 + 2X_2 \exp Y_1$$

$$0 = Y_3 - \sqrt{X_3}$$

The code input is the same as in Example 7.1 except as follows:

```
formal_model
  1 0 2   1 0 0   1
  1 1 0   0 1 0   0
  0 0 1   0 0 1   0
```

The code output is as follows:

List of the equations in each submodel:

Submodel 1:

E3 is solved by this submodel.

Submodel 2:

E1 is solved by this submodel.

E3 is part of this submodel but was previously solved.

Submodel 3:

E1 is part of this submodel but was previously solved.

E2 is solved by this submodel.

E3 is part of this submodel but was previously solved.

Paths to output 1 Y1

Y1 depends through submodel 2 on Y3 (depth= 1)

Y1 depends through submodel 2 on Y3

which depends through submodel 1 on X3 (depth= 2)

Y1 depends through submodel 2 on X1 (depth= 1)

Y1 depends through submodel 2 on P1 (depth= 1)

Paths to output 2 Y2

Y2 depends through submodel 3 on Y1 (depth= 1)

Y2 depends through submodel 3 on Y1

which depends through submodel 2 on Y3 (depth= 2)

Y2 depends through submodel 3 on Y1

which depends through submodel 2 on Y3
which depends through submodel 1 on X3 (depth= 3)

Y2 depends through submodel 3 on Y1

which depends through submodel 2 on X1 (depth= 2)

Y2 depends through submodel 3 on Y1

	which depends through submodel	2 on P1	(depth= 2)
Y2	depends through submodel	3 on X2	(depth= 1)
Paths to output	3	Y3	
Y3	depends through submodel	1 on X3	(depth= 1)

Net complexity measures:

Sequentiality	:	1
Formal model complexity	:	10
Full model complexity	:	21
Indirectness	:	2.10

From the list of submodels, it can be seen that the algorithm recognizes that it can solve for Y_3 first (submodel 1), then use the result to solve for Y_1 (submodel 2), then solve for Y_2 (submodel 3). The sequentiality is still ideal ($\sigma = 1$) because each submodel solves only one equation, even though submodels 2 and 3 contain multiple equations.

The pathway from X_3 to Y_2 has depth $\delta = 3$ because the dependence goes through intermediate steps. Someone attempting to understand the dependence of Y_2 on X_3 would have to “dig deeper” into the model to reveal it than if the dependence were explicit, hence the term *depth*.

7.3 Example 3

Now consider a more “dense” model using the same variables as in Examples 1 and 2:

$$0 = Y_1 Y_2 + 5X_1/Y_3 + Y_3 + X_2 X_3 + P_1$$

$$0 = Y_2 + 2X_2 \exp Y_1 - Y_3$$

$$0 = Y_1 Y_3 - \sqrt{X_3}$$

The code input is the same as in Example 7.1 except as follows:

```
formal_model
  1 1 2   1 1 1  1
  1 1 1   0 1 0  0
  1 0 1   0 0 1  0
```

The code output is as follows:

List of the equations in each submodel:

Submodel 1:

E1 is solved by this submodel.

E2 is solved by this submodel.

E3 is solved by this submodel.

Paths to output 1 Y1

Y1 depends through submodel 1 on Y2 (depth= 3)

Y1 depends through submodel 1 on Y3 (depth= 3)

Y1 depends through submodel 1 on X1 (depth= 3)

Y1 depends through submodel 1 on X2 (depth= 3)

Y1 depends through submodel 1 on X3 (depth= 3)

Y1 depends through submodel 1 on P1 (depth= 3)

Paths to output 2 Y2

Y2 depends through submodel 1 on Y1 (depth= 3)

Y2 depends through submodel 1 on Y3 (depth= 3)

Y2 depends through submodel 1 on X1 (depth= 3)

Y2 depends through submodel 1 on X2 (depth= 3)

Y2 depends through submodel 1 on X3 (depth= 3)

Y2 depends through submodel 1 on P1 (depth= 3)

Paths to output 3 Y3

Y3 depends through submodel 1 on Y1 (depth= 3)

Y3 depends through submodel 1 on Y2 (depth= 3)

Y3 depends through submodel 1 on X1 (depth= 3)

Y3 depends through submodel 1 on X2 (depth= 3)

Y3 depends through submodel 1 on X3 (depth= 3)

Y3 depends through submodel 1 on P1 (depth= 3)

Net complexity measures:

Sequentiality	:	3
Formal model complexity	:	15
Full model complexity	:	58
Indirectness	:	3.87

There is only one submodel, because all three equations must be solved simultaneously. The model is therefore not ideally sequential ($\sigma > 1$). As shown by the pathways, all the outputs depend on all the inputs, on the parameter, and the other outputs. The fairly large value of indirectness in this example reflects the fact that in this model, nearly “everything depends on everything,” so we anticipate difficulties in finding the source of any bad values that might occur in a finite element simulation.

7.4 Example 4

Using the same variables as in the previous examples, consider a model that contains multiple branches:

$$\begin{aligned}0 &= Y_1 + X_1 + P_1 \\0 &= Y_2 + X_2 \\0 &= \begin{cases} Y_1 + Y_3 & \text{if } X_3 > 0, \\ Y_2 + Y_3 & \text{otherwise.} \end{cases}\end{aligned}$$

To account for the condition, define an intermediate variable I_1 by

$$I_1 = \text{eval}\{X_3 > 0\}$$

then restate the model as

$$\begin{aligned}0 &= Y_1 + X_1 + P_1 \\0 &= Y_2 + X_2 \\0 &= I_1 - \text{eval}\{X_3 > 0\} \\0 &= I_1(Y_1 + Y_3) + (1 - I_1)(Y_2 + Y_3).\end{aligned}$$

The variables are now

$$\langle Y_1 \ Y_2 \ Y_3 \ I_1 \ X_1 \ X_2 \ X_3 \ P_1 \rangle$$

where I_1 , since it is an unknown until it is computed, is treated as an output.

The code input is as follows:

```
number_of_outputs
  4
outputs
  Y1
  Y2
  Y3
  I1
number_of_inputs
  3
inputs
  X1
  X2
  X3
number_of_parameters
  1
parameters
  P1
number_of_equations
  4
equations
```

```

E1
E2
E3
E4
formal_model
 1 0 0 0   1 0 0  1
 0 1 0 0   0 1 0  0
 0 0 0 1   0 0 1  0
 1 1 2 2   0 0 0  0

```

The code output is as follows:

List of the equations in each submodel:

```

Submodel  1:
  E1      is solved by this submodel.
Submodel  2:
  E2      is solved by this submodel.
Submodel  3:
  E3      is solved by this submodel.
Submodel  4:
  E1      is part of this submodel but was previously solved.
  E2      is part of this submodel but was previously solved.
  E3      is part of this submodel but was previously solved.
  E4      is solved by this submodel.

```

```

Paths to output  1  Y1
Y1      depends through submodel  1 on X1      (depth=  1)
Y1      depends through submodel  1 on P1      (depth=  1)
Paths to output  2  Y2
Y2      depends through submodel  2 on X2      (depth=  1)
Paths to output  3  Y3
Y3      depends through submodel  4 on Y1      (depth=  1)
Y3      depends through submodel  4 on Y1
  which depends through submodel  1 on X1      (depth=  2)
Y3      depends through submodel  4 on Y1
  which depends through submodel  1 on P1      (depth=  2)
Y3      depends through submodel  4 on Y2      (depth=  1)
Y3      depends through submodel  4 on Y2
  which depends through submodel  2 on X2      (depth=  2)
Y3      depends through submodel  4 on I1      (depth=  1)
Y3      depends through submodel  4 on I1
  which depends through submodel  3 on X3      (depth=  2)
Paths to output  4  I1
I1      depends through submodel  3 on X3      (depth=  1)

```

Net complexity measures:

Sequentiality	:	1
Formal model complexity	:	13
Full model complexity	:	21
Indirectness	:	1.62

The use of conditional statements in a model tends to increase the complexity metrics.

7.5 Example 5: Mooney-Rivlin model

The following is a statement of the Mooney-Rivlin model for rubber elasticity taken verbatim from the LAME material library documentation [3]:

$$\begin{aligned}
 U &= C_{10}(\bar{I}_1 - 3) + C_{01}(\bar{I}_2 - 3) + K(J_m \ln J_m - J_m) \\
 [\bar{B}] &= [\bar{F}] [\bar{F}]^T \\
 [\bar{F}] &= J^{-1/3} [F] \\
 J &= \det [F] \\
 \bar{I}_1 &= [I] : [\bar{B}] \\
 \bar{I}_2 &= \frac{1}{2} \left(\bar{I}_1^2 - [I] : ([\bar{B}] [\bar{B}]) \right) \\
 J_m &= J / J_{th} \\
 [F] &= [F_m] [F_{th}] \\
 [F_{th}] &= J_{th}^{1/3} [I] \\
 [\sigma'] &= \frac{2}{J_m} \text{dev} \left[(C_{10} + \bar{I}_1 C_{01}) [\bar{B}] - C_{01} [\bar{B}] [\bar{B}] \right] \\
 p &= K \ln J_m
 \end{aligned}$$

The inputs are:

$$[F], J_{th}.$$

The parameters are:

$$C_{10}, C_{01}, K.$$

The outputs, including intermediate quantities, are:

$$U, [\sigma'], p, \bar{I}_1, \bar{I}_2, [\bar{F}], [F_m], [F_{th}], J, J_m, [\bar{B}].$$

The symbols $[I]$, dev , and \det are considered to be standard tools of algebra and are not treated as variables. The full list of variables is as follows:

$$\langle v \rangle = \langle U \ [\sigma'] \ p \ \bar{I}_1 \ \bar{I}_2 \ [\bar{F}] \ [F_m] \ [F_{th}] \ J \ J_m \ [\bar{B}] \ [F] \ J_{th} \ C_{10} \ C_{01} \ K \rangle.$$

The ComplexityMetrics input is as follows:

```

number_of_outputs
  11
outputs
  U
  SigmaDev
  p

```

```

Ibar1
Ibar2
Fbar
Fm
Fth
J
Jm
Bbar
number_of_inputs
  2
inputs
  F
  Jth
number_of_parameters
  3
parameters
  C10
  C01
  K
number_of_equations
  11
equations
  E1
  E2
  E3
  E4
  E5
  E6
  E7
  E8
  E9
  E10
  E11
* the following line is a comment to help arrange the formal_model:
* U SigmaDev p Ibar1 Ibar2 Fbar Fm Fth J Jm Bbar      F Jth      C10 C01 K
formal_model
  1 0      0 1      1      0      0 0      0 3 0      0 0      1 1 1
  0 0      0 0      0      2      0 0      0 0 1      0 0      0 0 0
  0 0      0 0      0      1      0 0      1 0 0      1 0      0 0 0
  0 0      0 0      0      0      0 0      1 0 0      1 0      0 0 0
  0 0      0 1      0      0      0 0      0 0 1      0 0      0 0 0
  0 0      0 1      1      0      0 0      0 0 2      0 0      0 0 0
  0 0      0 0      0      0      0 0      1 1 0      0 1      0 0 0
  0 0      0 0      0      1      1 1      0 0 0      0 0      0 0 0
  0 0      0 0      0      0      0 1      0 0 0      0 1      0 0 0

```

0 1	0 1	0	0	0 0	0 1	3	0 0	1	2	0
0 0	1 0	0	0	0 0	0 1	0	0 0	0	0	1

The code output is as follows (only the pathways to U are included here):

List of the equations in each submodel:

Submodel 1:
 E9 is solved by this submodel.

Submodel 2:
 E4 is solved by this submodel.

Submodel 3:
 E3 is solved by this submodel.
 E4 is part of this submodel but was previously solved.

Submodel 4:
 E4 is part of this submodel but was previously solved.
 E7 is solved by this submodel.

Submodel 5:
 E4 is part of this submodel but was previously solved.
 E7 is part of this submodel but was previously solved.
 E11 is solved by this submodel.

Submodel 6:
 E2 is solved by this submodel.
 E3 is part of this submodel but was previously solved.
 E4 is part of this submodel but was previously solved.

Submodel 7:
 E2 is part of this submodel but was previously solved.
 E3 is part of this submodel but was previously solved.
 E4 is part of this submodel but was previously solved.
 E5 is solved by this submodel.

Submodel 8:
 E3 is part of this submodel but was previously solved.
 E4 is part of this submodel but was previously solved.
 E8 is solved by this submodel.
 E9 is part of this submodel but was previously solved.

Submodel 9:
 E2 is part of this submodel but was previously solved.
 E3 is part of this submodel but was previously solved.
 E4 is part of this submodel but was previously solved.
 E5 is part of this submodel but was previously solved.
 E6 is solved by this submodel.

Submodel 10:
 E2 is part of this submodel but was previously solved.
 E3 is part of this submodel but was previously solved.

E4 is part of this submodel but was previously solved.
 E5 is part of this submodel but was previously solved.
 E7 is part of this submodel but was previously solved.
 E10 is solved by this submodel.

Submodel 11:

E1 is solved by this submodel.
 E2 is part of this submodel but was previously solved.
 E3 is part of this submodel but was previously solved.
 E4 is part of this submodel but was previously solved.
 E5 is part of this submodel but was previously solved.
 E6 is part of this submodel but was previously solved.
 E7 is part of this submodel but was previously solved.

Paths to output 1 U

U depends through submodel 11 on Ibar1 (depth= 1)
 U depends through submodel 11 on Ibar1
 which depends through submodel 7 on Bbar (depth= 2)
 U depends through submodel 11 on Ibar1
 which depends through submodel 7 on Bbar
 which depends through submodel 6 on Fbar (depth= 3)
 U depends through submodel 11 on Ibar1
 which depends through submodel 7 on Bbar
 which depends through submodel 6 on Fbar
 which depends through submodel 3 on J (depth= 4)
 U depends through submodel 11 on Ibar1
 which depends through submodel 7 on Bbar
 which depends through submodel 6 on Fbar
 which depends through submodel 3 on J
 which depends through submodel 2 on F (depth= 5)
 U depends through submodel 11 on Ibar1
 which depends through submodel 7 on Bbar
 which depends through submodel 6 on Fbar
 which depends through submodel 3 on F (depth= 4)
 U depends through submodel 11 on Ibar2 (depth= 1)
 U depends through submodel 11 on Ibar2
 which depends through submodel 9 on Ibar1 (depth= 2)
 U depends through submodel 11 on Ibar2
 which depends through submodel 9 on Ibar1
 which depends through submodel 7 on Bbar (depth= 3)
 U depends through submodel 11 on Ibar2
 which depends through submodel 9 on Ibar1
 which depends through submodel 7 on Bbar
 which depends through submodel 6 on Fbar (depth= 4)
 U depends through submodel 11 on Ibar2
 which depends through submodel 9 on Ibar1

	which depends through submodel	7 on Bbar	
	which depends through submodel	6 on Fbar	
	which depends through submodel	3 on J	(depth= 5)
U	depends through submodel	11 on Ibar2	
	which depends through submodel	9 on Ibar1	
	which depends through submodel	7 on Bbar	
	which depends through submodel	6 on Fbar	
	which depends through submodel	3 on J	
	which depends through submodel	2 on F	(depth= 6)
U	depends through submodel	11 on Ibar2	
	which depends through submodel	9 on Ibar1	
	which depends through submodel	7 on Bbar	
	which depends through submodel	6 on Fbar	
	which depends through submodel	3 on F	(depth= 5)
U	depends through submodel	11 on Ibar2	
	which depends through submodel	9 on Bbar	(depth= 2)
U	depends through submodel	11 on Ibar2	
	which depends through submodel	9 on Bbar	
	which depends through submodel	6 on Fbar	(depth= 3)
U	depends through submodel	11 on Ibar2	
	which depends through submodel	9 on Bbar	
	which depends through submodel	6 on Fbar	
	which depends through submodel	3 on J	(depth= 4)
U	depends through submodel	11 on Ibar2	
	which depends through submodel	9 on Bbar	
	which depends through submodel	6 on Fbar	
	which depends through submodel	3 on J	
	which depends through submodel	2 on F	(depth= 5)
U	depends through submodel	11 on Ibar2	
	which depends through submodel	9 on Bbar	
	which depends through submodel	6 on Fbar	
	which depends through submodel	3 on F	(depth= 4)
U	depends through submodel	11 on Jm	(depth= 1)
U	depends through submodel	11 on Jm	
	which depends through submodel	4 on J	(depth= 2)
U	depends through submodel	11 on Jm	
	which depends through submodel	4 on J	
	which depends through submodel	2 on F	(depth= 3)
U	depends through submodel	11 on Jm	
	which depends through submodel	4 on Jth	(depth= 2)
U	depends through submodel	11 on C10	(depth= 1)
U	depends through submodel	11 on C01	(depth= 1)
U	depends through submodel	11 on K	(depth= 1)

Net complexity measures:

Sequentiality	:	1
Formal model complexity	:	43
Full model complexity	:	217
Indirectness	:	5.05

In the listing of the submodels, note the progression from submodels with fewer equations to those with more.

The Mooney-Rivlin model as described in the LAME documentation is found by the code to be ideally sequential. However, there are 7 pathways that start with $[F]$ and end with U , revealing that the dependence of U on $[F]$ is somewhat concealed by the heavy use of intermediate quantities. This is reflected by the large value of indirectness shown at the end of the model output.

If temperature dependence were not included, all of the complexity metrics would be substantially reduced. This is an example of the increase in complexity of models when multiple physical fields are included (“multiphysics” models).

8 Conclusions

A fundamental principle in this work is that dependence between variables has *direction*, that is, the fact that y_2 depends on y_1 does not imply that y_1 depends on y_2 . Recognizing this principle immediately reveals the inadequacy of superficial measures, such as merely counting the number of material parameters N_p , to fully characterize the complexity of a model.

The complexity metrics proposed here are an attempt to reduce the subjectivity in comparing the complexity of different material models. By design, the metrics avoid any reliance on understanding of the physics, since this varies between individuals. The metrics also avoid actually running calculations with the model. This increases their potential usefulness during the development phase, rather than the software implementation phase, at which time may be too late for such metrics to have any benefit.

It may be seen from Example 5, the Mooney-Rivlin model, that a large number of variables does not necessarily reduce the sequentiality of a model, making it relatively straightforward to implement. It would also be possible to have a model with a large number of inputs, outputs, and parameters that is easy to understand if the indirectness is low.

However, the overall trend in the examples, including the Mooney-Rivlin model, is that when intermediate variables are used, and when outputs depend on other outputs, there is a snowballing effect on the full complexity and indirectness metrics as the number of variables increases. This trend of the full complexity outpacing the formal complexity as variables are added is a manifestation of the “out-of-control” complexity of models that is sometimes perceived when more and more ingredients, such as multiphysical fields and rate dependence, are added to a model. The inclusion of temperature in the Mooney-Rivlin mechanical model is a benign but illustrative example of this.

References

- [1] J. P. Kearney, R. L. Sedlmeyer, W. B. Thompson, M. A. Gray, and M. A. Adler, Software complexity measurement, *Communications of the ACM* **29** (1986) 1044–1050.
- [2] D. J. Spiegelhalter, N. G. Best, B. P. Carlin, and A. Van Der Linde, Bayesian measures of model complexity and fit, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **64** (2002) 583–639.
- [3] W. M. Scherzinger and D. C. Hammerand, Constitutive models in LAME, technical report SAND2007-5873, Sandia National Laboratories (2007).

DISTRIBUTION:

- 1 MS 0899 Technical Library, 9536 (electronic copy)



Sandia National Laboratories