Australian Government
Department of Defence
Defence Science and
Technology Organisation

Vector and Matrix Variable Definition in DAVE-ML

Prepared by: Mr Geoff Brian[1],
Defence Science and Technology Organisation,
Department of Defence, Australia.
<Geoff.Brian@dsto.defence.gov.au>

Abstract

The Modeling and Simulation Technical Committee of the American Institute of Astronautics and Aeronautics (AIAA) are developing a series of standards to facilitate the exchange of aircraft dynamics simulations models between aircraft simulations agencies and research establishments. One of these standards has been developed for the exchange of aircraft characteristic data – BSR/AIAA S-119-2010 Flight Dynamics Model Exchange Standard, [BSR/AIAA S-119-2010]. The standard defines procedures and formats for the encapsulation of aircraft characteristics such as aerodynamic, propulsion, and mass property data.

The syntax for encapsulating data is currently limited to scalar parameters. This document defines an extension to the encapsulation syntax enabling the management of data referenced as vectors or n-dimensional matrices. The attributes and dependencies of modified and additional DAVE-ML elements needed to support vectors and matrices are discussed. Examples of encapsulating data as vectors and matrices are provided.

1 Mr Brian is an employee of the Australian Department of Defence, Defence Science and Technology Organisation. The work documented in this report was conducted while on attachment to NASA Langley Research Center under a Visiting Research agreement with the National Institute of Aerospace.

Table of Contents

Abbreviations

AIAA	American Institute of Aeronautics and Astronautics
DAVE-ML	Dynamic Aerospace Vehicle Exchange Markup Language
DSTO	Defence Science and Technology Organisation, Australia
DTD	Document Type Definition
MathML	Mathematical Markup Language
NASA LaRC	National Aeronautics and Space Administration, Langley Research Center
XML	eXtensible Markup Language
W3C	World Wide Web Consortium

Acknowledgements

I wish to acknowledge the valuable comments provided in correspondence with Dennis Linse, Bruce Jackson (NASA LaRC), Shane Hill (DSTO), Dan Newman (Quantitative Aeronautics) and Rob Curtin (Qinetiq) in formulating the concepts that are instantiated in this document.

1 The Exchange of Aircraft Models and Data

The Modeling and Simulation Technical Committee of the American Institute of Astronautics and Aeronautics (AIAA) are developing a series of standards to facilitate the exchange of aircraft dynamics simulations models between aircraft simulations agencies and research establishments. One of these standards has been developed for the exchange of aircraft characteristic data – BSR/AIAA S-119-2010 Flight Dynamics Model Exchange Standard, [BSR/AIAA S-119-2010]. The standard defines procedures and formats for the encapsulation of aircraft characteristics such as aerodynamic, propulsion, and mass property data.

The objectification of this standard is the Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML), which defines a syntactical language for encoding model data, [DAVE-ML]. DAVE-ML uses a text-based format built upon the eXtensible Markup Language (XML) Version 1.1 and Mathematical Markup Language (MathML) Version 2.0 open standards developed by the World Wide Web Consortium (W3C). It defines additional grammar to provide a domain-specific language for aerospace flight dynamics modelling, verification, and documentation, utilising a Document Type Definition (DTD). The DAVE-ML DTD defines the rules of the markup language syntax, and is used to verify that a compliant file is well formed.

Version 2.0 RC3.0 of the DAVE-ML syntax is limited to encapsulating data as scalar parameters. This document details an extension to the syntax enabling the management of data referenced as vectors or n-dimensional matrices. The attributes and dependencies of modified and additional DAVE-ML grammar elements required to support vectors and matrices are discussed. Examples of encapsulating data as vectors and matrices are provided.

The syntax supporting vectors and matrices, in itself, does not define the *best practice* rules for interacting with vectors and matrices; for example, how the values in a vector or matrix should be set if only provided with a single scalar value. Recommendations for these *best practice* rules are provided for guidance, however, they are not enforced by the syntax.

The management of data as a vector or matrix should not be thought of as a alternative to the storage of data in a gridded table, where typically large quantities of data are stored. In contrast, vectors and matrices for flight dynamic applications are generally small in size and used directly in mathematical expressions for solving problems such as a vehicle's equations of motion. By way of an example, discrete data for the lift coefficient of a vehicle should be stored in either a gridded or ungridded table, while data for the inertia tensor could be represented by a matrix.

2 MathML Vectors and Matrices

The World Wide Web Consortium (W3C) have undertaken a project to developed a syntax for encoding mathematical expressions that are suitable for scientific communication of both notation and meaning. This syntax is known as the Mathematical Markup Language (MathML), [MathML]. Reference [MathML] discusses the history and purpose of MathML, together with the syntax and examples.

DAVE-ML includes the capability to define equations representing data that may be of interest for aircraft modelling and simulation activities. This capability is provided using the MathML syntax for expressing mathematical equations. A simple example of this syntax is:

$$C_D = C_{D_o} + 0.7\,C_L^2$$

```
<calculation>
  <math>
    <apply>
      <plus/>
      <ci>baseDragCoefficient</ci>
      <apply>
        <times/>
        <cn>0.7</cn>
        <ci>liftCoefficient</ci>
        <ci>liftCoefficient</ci>
      </apply>
    </apply>
  </math>
</calculation>
```

To date the adoption of the MathML syntax within the DAVE-ML framework has been limited to scalar variables. The extension of DAVE-ML to support vector and matrix variables requires the development of a suitable syntax to manage the population of these variables, together with a syntax enabling algebraic operations to be defined.

MathML has within its syntax definitions to support vector and matrix operations[2]. Vectors and matrices are defined using the `vector` and `matrix` tags respectively. A vector is defined as a single dimensional matrix, while a matrix is a two dimensional array of parameters, [MathML]. It is possible to define higher order dimensional matrices through the combined use of vectors and matrices, for example:

- a 3-D matrix is a vector of 2-D matrices,
- a 4-D matrix is a 2-D matrix or 2-D matrices,
- a 5-D matrix is a vector of 4-D matrices,
- and so forth.

In addition, vector and matrix operators defined within MathML (in addition to normal arithmetic

2 Mathematical tools such as Matlab®, Octave and Matrix X®, are also script based applications that provide syntax for operating with vectors and matrices. However, their syntax is either proprietary or not compatible with the mark-up framework of XML, on which DAVE-ML is based.

operators such as add, subtract, multiply, …) include `transpose`, `inverse`, `determinant`, `vectorproduct`[3], `scalarproduct`[4], `outerproduct`[5], and `selector` - which enables elements or rows of a vector or matrix to be extracted. Unfortunately operators to compute the trace, adjoint and rank of matrices are not currently included in MathML.

Defining n-dimensional matrices using the MathML syntax is cumbersome. Furthermore, MathML defines subordinate elements for populating vector and matrices that again are cumbersome and not suited to the application of aircraft model and data exchange provided by DAVE-ML.

A syntax more suited to aircraft modelling and simulation applications is detailed in the following section for defining vectors and matrices in DAVE-ML compliant files. The MathML syntax for defining vector and matrix operations is retained.

3 The DAVE-ML variableDef Element

Vectors and matrices are in essence the same as scalar variables except that they represent a series of data instead of a singular value. Therefore, it is proposed to extend the DAVE-ML variable definition, `variableDef`, to embrace scalars, vectors and matrices.

In Version 2 RC3.0 of the DAVE-ML documentation, [DAVE-ML], the `variableDef` is defined as:

```
variableDef : name, varID, units, [axisSystem], [initialValue]
     description?
     (provenance | provenanceRef)?
     calculation?
     (isInput | isControl | isDisturbance)?
     isState?
     isStateDeriv?
     isOutput?
     isStdAIAA?
     uncertainty?
```

The attributes of the `variableDef` are:

name	The name of the signal being defined.
varID	An internal identifier for the signal.
units	The units of the signal.
axisSystem	(optional) The axis in which the signal is measured.
sign	(optional) The sign convention for the signal, if any.
alias	(optional) Possible alias name (facility specific) for the signal.
symbol	(optional) UNICODE symbol for the signal.
initialValue	(optional) An initial and possibly constant numeric value for the signal.

3 The MathML definition for `vectorproduct` is equivalent to that of the **cross product,** [Hubbard], [Kuo].

4 The MathML definition for `scalarproduct` is equivalent to that of the **dot product** , [Hubbard], [Kuo].

5 The `outerproduct` multiplies two vectors to form a matrix: $A = uv^T$, where u and v are vectors, [MathML].

while the sub-element properties are:

description	(optional) A text description of the variable description
provenance	(optional) Information on the history of the data
provenanceRef	(optional) A reference to provenance record
calculation	(optional) A Math-ML calculation
isInput	(optional) Identifies the variable as an input
isControl	(optional) Identifies the variable as a control
isDisturbance	(optional) Identifies the variable as a external disturbance
isState	(optional) Identifies the variable as a state variable
isStateDeriv	(optional) Identifies the variable as a state derivative
isOutput	(optional) Identifies the variable as an output
isStdAIAA	(optional) Indicates that the variable conforms with the AIAA exchange standard, [BSR/AIAA S-119].
uncertainty	(optional) Describes the uncertainty of the parameter.

Embracing vectors and matrices using the variable definition requires additional sub-elements to be included in order to specify the dimensions the vector or matrix, together with defining the entries of the vector or matrix. These additional elements are discussed in the following section.

Variables defined in a DAVE-ML compliant XML file may be either *set* or *requested* by an external application. In addition, a variable may be used internally to evaluate another variable. These properties are also inherited by a vector or matrix variable. However, the fact that vectors and matrices have more than one entry increases the level of complexity when defining a suitable syntax for interacting with the variable.

4 Additions to variableDef Element

Support for vectors and matrices could be achieved by including the sub-elements highlighted in bold text to the variable definition.

```
variableDef : name, varID, units, [axisSystem], [initialValue]
     description?
     (provenance | provenanceRef)?
     dimensionDef?
     (calculation | array)?
     (isInput | isControl | isDisturbance)?
     isState?
     isStateDeriv?
     isOutput?
     isStdAIAA?
     uncertainty?
```

where:

array	The data for the vector or matrix.
dimensionDef	A list of dimensions (dim) for the vector or matrix

The list of dimensions defined using the `dimensionDef` element specifies either the number of rows of data in a vector, or the size of each dimension in a n-dimensional matrix. The order for defining the size of each dimension of a matrix starts with the number of columns in the base matrix, followed by the number of rows of the base matrix, and then the number of base matrices making up the third dimension, and so forth for n-dimensional matrices. For example:

- for a vector with 3 entries:

```
<dimensionDef> <!-- Number of columns is 1, therefore not included -->
  <dim>3</dim> <!-- Number of rows in the vector -->
</dimensionDef>
```

- for a 2 dimensional matrix:

```
<dimensionDef>
  <dim>3</dim> <!-- Number of columns in the matrix -->
  <dim>3</dim> <!-- Number of rows in the matrix -->
</dimensionDef>
```

- for a 3 dimensional matrix:

```
<dimensionDef>
  <dim>3</dim> <!-- Number of columns in the base matrix -->
  <dim>3</dim> <!-- Number of rows in the base matrix -->
  <dim>2</dim> <!-- Number of base matrices comprising 3rd dimension -->
</dimensionDef>
```

The data for a vector or matrix is specified using the `array` element. This is in effect a table of data, and thus utilises the `dataTable` element from DAVE-ML for encoding the vector and matrix entries. The entries may be either numeric values or references to other variable definitions. The entries for a vector represent the row entries of that vector. The entries for a matrix are specified such that the column entries of the 1st row are listed followed by column entries for subsequent rows until the base matrix is complete. This sequence is repeated for higher order matrix dimensions until all entries of the matrix are specified. The following examples illustrate this procedure:

- for a vector with 3 entries

```
<array>
  <dataTable>
    0.0,
    eulerInclinationAngle_,
    eulerRollAngle_
  </dataTable>
</array>
```

- for a 2 dimensional matrix

```
<array>
  <dataTable>
    <!-- Row #1  IXX, -IXY, -IXZ -->
    inertiaIXX, -inertiaIXY, -inertiaIXZ,
    <!-- Row #2 -IXY,  IYY, -IYZ -->
    -inertiaIXY, inertiaIYY, -inertiaIYZ,
    <!-- Row #3 -IXZ, -IYZ,  IZZ -->
    -inertiaIXZ, -inertiaIYZ, inertiaIZZ,
  </dataTable>
</array>
```

- for a 3 dimensional matrix

```
<array>
  <dataTable>
    <!-- First 3x3 Matrix -->
    <!-- Row #1 -->
    1.0, 0.0, 0.0,
    <!-- Row #2 -->
    0.0, 1.0, 0.0,
    <!-- Row #3 -->
    0.0, 0.0, 1.0,
    <!-- Second 3x3 Matrix -->
    <!-- Row #1  IXX, -IXY, -IXZ -->
    inertiaIXX, -inertiaIXY, -inertiaIXZ,
    <!-- Row #2 -IXY,  IYY, -IYZ -->
    -inertiaIXY, inertiaIYY, -inertiaIYZ,
    <!-- Row #3 -IXZ, -IYZ,  IZZ -->
    -inertiaIXZ, -inertiaIYZ, inertiaIZZ,
  </dataTable>
</array>
```

There is also a case for referencing vector and matrices, and components of, by variables that are evaluated using MathML calculations. The nuances of interacting with the vectors and matrices from within a MathML calculation are discussed in Section 6.

Section 8 provides more vector and matrix variable definition examples, together with examples of examples of vectors and matrices being used in MathML calculations.

5 Supplementary Elements

Although the `dimensionDef` and `array` elements discussed above permit the encoding of vectors and matrices through the variable definition, they are dependent on sub-elements to achieve this. The `dimensionDef` uses a sub-element of `dim` to define the size of each dimension of the vector or matrix, while `array` uses the `dataTable` sub-element to encode the entries of the vector or matrix. Therefore, it is necessary to add definitions for these elements in the DAVE-ML DTD.

The definition of the `dimensionDef` element to be included in the DAVE-ML DTD would be:

```
<!ELEMENT dimensionDef (dim+)>
```

while that for the `dim` element is:

```
<!ELEMENT dim (#PCDATA)>
```

The definition of the `array` element is:

```
<!ELEMENT array (dataTable)>
```

The `dataTable` element already exists in the DAVE-ML DTD, being used for encoding gridded table information.

6 Vector and Matrices in MathML Calculations

With any vector or matrix capability it is ideal to have a syntax that permits typical vector and matrix calculations to be performed thereby creating new variables. As discussed in Section 2, MathML provides a suitable syntax for defining mathematical operations for scalars, vectors and matrices. Operators such as `plus`, `times`, and `minus` apply equally to scalars, vectors and matrices and may be used when mixing variables types. The `transpose`, `inverse`, `determinant`, `vectorproduct`, `scalarproduct`, `outerproduct`, and `selector` operators are only used with vector and matrix variables types.

The DAVE-ML variable definition has been structured so that a variable may be defined using an equation instead of data. This is achieved by encoding the equation using the `calculation` sub-element. The following is a simple example of defining a variable that is calculated from an equation:

```
<variableDef name="dragCoefficient_nd" varID="dragCoefficient" units="nd">
 <description> Drag Coefficient</description>
 <calculation>
   <math>
     <apply>
       <plus/>
       <ci>baseDragCoefficient</ci>
       <apply>
         <times/>
         <cn>0.7</cn>
         <ci>liftCoefficient</ci>
         <ci>liftCoefficient</ci>
       </apply>
     </apply>
   </math>
 </calculation>
 <isOutput/>
</variableDef>
```

Using equations to calculate variables involving vectors and matrices is essentially the same. However, it is necessary to defined the size of the vector or matrix that results from the calculation. This is not required if a scalar value is the result. By the way of an example, if a matrix **M** has two dimensions, with two rows and three columns, and a vector **V** has three entries, then multiplying the matrix by the vector will produce a vector **R** with two entries.

$$R = [M]V$$

$$\begin{bmatrix} \square \\ \square \end{bmatrix} = \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \end{bmatrix} \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix}$$

The resulting variable definition for R is:

```
<variableDef name="R" varID="R" units="">
 <description> Multiplying a matrix and vector</description>
 <dimensionDef>
   <dim>2</dim> <!-- Number of entries in the output vector -->
 </dimensionDef>
 <calculation>
   <math>
     <apply>
       <times/>
       <ci>M</ci>
       <ci>V</ci>
     </apply>
   </math>
 </calculation>
 <isOutput/>
</variableDef>
```

As discussed in Section 2, the MathML syntax for defining vector and matrix operators has been adopted, but the syntax for defining vector and matrix variables has not. As a consequence, the procedures for using these operators differs from examples documented in [MathML]. In most cases the differences are minimal, and typically related to not specifying the `type` attribute of the working variable. The following discussion details the procedures for using the MathML vector and matrix operators in the context of defining vector and matrix variables as DAVE-ML `variableDef`s. The differences in implementation compared with [MathML] are also highlighted. Detailed examples of using the vector and matrix MathML syntax are provided in Section 8.

6.1 Transpose

The operation of transposing a vector or matrix may be defined using the `transpose` operator. It applies only to one vector or matrix, defined using a reference to a predefined `variableDef`. The syntax for the `transpose` operator is:

```
<transpose/>
  <ci>vectorOrMatrix</ci>
```

The difference between this usage and that in [MathML] is that the `type` attribute of the working variable is not required.

6.2 Inverse

The operation of computing the inverse of a matrix is achieved using the `inverse` operator. MathML uses this operator to also specifying the inverse of a function. When used for matrix operations it applies only to one matrix, defined using a reference to a predefined `variableDef`. The syntax for the `inverse` operator is:

```
<inverse/>
  <ci>matrix</ci>
```

The difference between this usage and that in [MathML] is that the `type` attribute of the working variable is not required.

6.3 Determinant

The operation of computing the determinant of a matrix may be defined using the `determinant` operator. The matrix must be square to compute the determinant. The result is a scalar. It applies only to one matrix, defined using a reference to a predefined `variableDef`. The syntax for the `determinant` operator is:

```
<determinant/>
  <ci>matrix</ci>
```

The difference between this usage and that in [MathML] is that the `type` attribute of the working variable is not required.

6.4 Scalarproduct

The operation of computing the scalar product of two vectors may be defined using the `scalarproduct` operator. The scalar product is equivalent to that of the *dot product* defined in texts dealing with linear algebra, [Hubbard], [Kuo]. The two vectors must have the same number of entries. The result is a scalar. The two working vectors are defined using references to predefined `variableDef`s. The syntax for the `scalarproduct` operator is:

```
<scalarproduct/>
  <ci>vector1</ci>
  <ci>vector2</ci>
```

There is no difference between this usage and that in [MathML].

6.5 Vectorproduct

The operation of computing the vector product of two vectors may be defined using the `vectorproduct` operator. The vector product is equivalent to that of the *cross product* defined in texts dealing with linear algebra, [Hubbard], [Kuo]. Each of the two vectors must have three entries. The result is a vector of size three. The two working vectors are defined using references to predefined `variableDef`s. The syntax for the `vectorproduct` operator is:

```
<vectorproduct/>
  <ci>vector1</ci>
  <ci>vector2</ci>
```

There is no difference between this usage and that in [MathML].

6.6 Outerproduct

The operation of computing the outer product of two vectors may be defined using the `outerproduct` operator. The outer product multiplies two vectors to form a matrix: $A = uv^T$, where u and v are vectors, [MathML]. The result is a matrix with size:

number of rows = number of rows of u, and
number of columns = number of rows of v.

The two working vectors are defined using references to a predefined `variableDef`s. The syntax for the `outerproduct` operator is:

```
<outerproduct/>
  <ci>vector1</ci>
  <ci>vector2</ci>
```

There is no difference between this usage and that in [MathML].

6.7 Selector

The `selector` operator permits the extraction of entries from vectors and matrices. As defined in [MathML] it has limited capability with application to lists, vectors and 2-D matrices. A row may be extracted from a matrix, but it does not appear to be suitable for extracting matrix columns, diagonals or sub-matrices, or working with matrices of higher than 2 dimensions.

The `selector` operator as defined for use with DAVE-ML extends the capability of the MathML operator by using the `other` attribute associated with `selector`. This attribute is used to specify the type of selection to be performed, being either:

element	to extract a single vector or matrix entry
row	to extract a row from a matrix
column	to extract a column from a matrix
diag	to extract a diagonal vector from a matrix
mslice	to extract a sub-matrix from a matrix

```
<selector other="row"/>
  <ci>argument_1</ci>
  <ci>argument_2</ci>
  <ci>...</ci>
```

The number of arguments that are associated with the `selector` operator depends on the choice of the `other` attribute, as well and the dimensions of the vector or matrix.

The first argument is the name of the vector or matrix from which data is to be extracted. The following arguments define the plane, row or column to be extracted, or where extraction starts. These can be either numeric values, specified using <cn> or references to scalar values using <ci>. MathML numbers rows rows and columns of vectors and matrices starting from a value of 1. This numbering convention has therefore been adopted here.

MathML defines the size of the argument list for the `selector` as *nary* - or effectively undefined. However, [MathML] states that a maximum of three arguments would be expected, including specification of the vector or matrix. Although this limitation is ignored here, where the maximum number of arguments is instead related to the dimension of the vector or matrix being processed, the argument list defined here still complies with MathML's `selector` definition.

6.7.1 element

This represents the process for extracting an entry from a vector or matrix. One argument is provided for each dimension of the vector or matrix in order to nominate the element to be extracted. The result is a scalar value.

For a 2-D matrix **M** extracting the entry from the 2nd row and 2nd column:

```
<selector other="element"/>
      <ci>M</ci>
      <!-- Need one argument for each dimension of the vector or matrix -->
      <cn>2</cn> <!-- Row number of entry -->
      <cn>2</cn> <!-- Column number of entry -->
```

6.7.2 row

This represents the process for extracting a row, or part of a row, from a matrix. An argument is provided for each plane (column) of the matrix if it is larger than 2 dimensional, together with an argument indicating the nominated row to be extracted. The value specifying the variable's dimension – through `dimensionDef` – indicates the number of elements to be extracted from the nominated row, staring from entry 1. The result is a vector.

For a 3-D matrix **M** extracting the 2nd row from the 2nd plane:

```
<selector other="row"/>
      <ci>M</ci>
      <!-- Need one argument for each plane of matrices having more
           than two dimensions -->
      <cn>2</cn>
      <cn>2</cn> <!-- Row number of extract -->
```

6.7.3 column

This represents the process for extracting a column, or part of a column, from a matrix. An argument is provided for each plane (column) of the matrix if it is larger than 2 dimensional, together with an argument indicating the nominated column to be extracted. The value specifying the variable's dimension – through `dimensionDef` – indicates the number of elements to be extracted from the nominated column, starting from entry 1. The result is a vector.

For a 3-D matrix **M** extracting the 1st column from the 2nd plane:

```
<selector other="column"/>
      <ci>M</ci>
      <!-- Need one argument for each plane of matrices having more
           than two dimensions -->
      <cn>2</cn>
      <cn>1</cn> <!-- Column number of extract -->
```

6.7.4 diag

This represents the process for extracting a planar diagonal, or there part of, from a matrix. An argument is provided for each plane (column) of the matrix if it is larger than 2 dimensional, together with an argument indicating the starting row number and an argument indicating the

starting column number. The value specifying the variable's dimension – through `dimensionDef` – indicates the number of elements to be extracted from the diagonal. The result is a vector.

For a 3-D matrix **M** extracting the diagonal starting at row 1, column 2 from the 2nd plane:

```
<selector other="diag"/>
      <ci>M</ci>
      <!-- Need one argument for each plane of matrices having more
           than two dimensions -->
      <cn>2</cn>
      <cn>1</cn> <!-- Row number to start extraction -->
      <cn>2</cn> <!-- Column number of start extraction -->
```

6.7.5 mslice

This represents the process for extracting a matrix from a matrix. The dimension of the extracted matrix is equal to or less than the dimension of the original matrix. The number of entries extracted in each dimension is specified through the variable's dimension element – `dimensionDef`. Arguments are specified nominating where extraction is to start, with an argument for each dimension of the original matrix.

For a 3-D matrix **M** extracting a 3-D matrix starting at row 1, column 2 from the 2nd plane:

```
<selector other="mslice"/>
      <ci>M</ci>
      <cn>2</cn> <!-- Plane number to start extraction -->
      <cn>1</cn> <!-- Row number to start extraction -->
      <cn>1</cn> <!-- Column number of start extraction -->
```

6.8 Extensions

An extension to the MathML syntax is required in order to support vector and matrix operations such as computing the trace, adjoint and the rank. Other operations that may be of interest include determining the eigenvectors and the associate eigenvalues of a matrix.

A syntax defining elements such as `trace`, `adjoint`, `rank`, `eigenvectors`, `eigenvalues` could be introduced through a DTD that inherits and extends the MathML syntax. Adding these elements to the DAVE-ML DTD could be an option, but it is preferable to keep the definition of mathematical operators separate from the definition of variables and data structures. Therefore, it is not recommended to include these extensions to the MathML syntax in DAVE-ML, instead a separated DTD tailored for mathematical operations should be created.

The development of a DTD extending the MathML syntax to include these vector and matrix operators is not presented here. Instead it is left as a future activity if this functionality is found to be required. Future versions of the MathML syntax may also included this capability, alleviating the need to develop a separate DTD.

7 Input Vector and Matrix Variables

This document is concerned primarily with defining a syntax that enables the management of data as vectors and matrices within the DAVE-ML framework. However, although not directly concerned with the syntax for defining vector and matrix variables, the following recommendations are provided as guidance for interacting with vector and matrix variables that are specified as *inputs*[6].

If a vector or matrix variable is defined as an input variable, then an external application should set the variable with consistent data. That is, a vector variable should be set using a vector of data, with an equivalent number of entries, and a matrix variable set using a matrix of data. A number of exceptions to this process are conceivable.

In the DAVE-ML syntax scalar input variables may be set using either the `initialValue` attribute of the `variableDef`, or be provided with a scalar value from an external application, which is stored for future reference. The purpose of the `initialValue` attribute should not change from one representing a scalar value to representing a sequence of data, as this capability can be provided through `array` sub-element. However, if the `array` sub-element is not used then it is recommended that all entries in the vector or matrix be set to the value of the `initialValue`. Similarly, if an external application sets the value for the vector or matrix using only a scalar, then it is recommended that all entries for the variable are set to that value.

Analogous to this is the setting of a matrix with a vector from an external application. It is recommended that the vector is assumed to represent the entries of a row of data for the matrix, and that all rows are set using the same data vector.

Setting high dimension matrices, greater than 2-D, using sub-matrices is not recommended.

6 In DAVE-ML an input variable is indicated using the `isInput` sub-element when defining the `variableDef`.

8 Examples

The following examples define vectors and matrices using the proposed syntax. In addition, examples applying vector and matrix operations to compute new variables are included.

8.1 Example Scalar Variable Definitions

These scalar variable definitions are included here as they are used during the definition of the vector and matrix examples.

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Scalar Variable Definitions: (used by vector and matrix definitions)
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
<!-- Inputs -->
<variableDef name="fuelMass_kg" varID="fuelMass_" units="kg"
  initialValue="1.0">
  <description> Represents a consumable fuel mass for the body</description>
</variableDef>

<variableDef name="eulerInclinationAngle_rad"
             varID="eulerInclinationAngle_"
             units="rad" initialValue="0.08726">
  <description> Represents the Euler inclination angle - Theta</description>
</variableDef>

<variableDef name="eulerRollAngle_rad"
             varID="eulerRollAngle_" units="rad" initialValue="0.52356">
  <description> Represents the Euler roll (bank) angle - Phi</description>
</variableDef>

<variableDef name="eulerAzimuthAngle_rad"
             varID="eulerAzimuthAngle_" units="rad" initialValue="0.052356">
  <description> Represents the Euler azimuth (yaw) angle - Psi</description>
</variableDef>

<!-- Internal -->
<variableDef name="inertiaIXX_kg.m2" varID="inertiaIXX" units="kg.m2">
  <description> This represents a bodies XX inertia</description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <apply>
          <times/>
          <cn>2000.0</cn>
          <ci>fuelMass_</ci>
        </apply>
        <cn>30000.0</cn>
      </apply>
    </math>
  </calculation>
</variableDef>
```

```
<variableDef name="inertiaIYY_kg.m2" varID="inertiaIYY" units="kg.m2">
  <description> This represents a bodies YY inertia</description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <apply>
          <times/>
          <cn>200.0</cn>
          <ci>fuelMass_</ci>
        </apply>
        <cn>3000.0</cn>
      </apply>
    </math>
  </calculation>
</variableDef>

<variableDef name="inertiaIZZ_kg.m2" varID="inertiaIZZ" units="kg.m2">
  <description> This represents a bodies ZZ inertia</description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <apply>
          <times/>
          <cn>400.0</cn>
          <ci>fuelMass_</ci>
        </apply>
        <cn>6000.0</cn>
      </apply>
    </math>
  </calculation>
</variableDef>

<variableDef name="inertiaIXZ_kg.m2" varID="inertiaIXZ" units="kg.m2"
   initialValue="540.0">
  <description> This represents a bodies XZ inertia</description>
</variableDef>

<variableDef name="inertiaIXY_kg.m2" varID="inertiaIXY" units="kg.m2"
   initialValue="650.0">
  <description> This represents a bodies XY inertia</description>
</variableDef>

<variableDef name="inertiaIYZ_kg.m2" varID="inertiaIYZ" units="kg.m2"
   initialValue="2300.0">
  <description> This represents a bodies YZ inertia</description>
</variableDef>
```

8.2 Examples of Vector Variable Definitions

The following are sample vector variable definitions. The first is a vector of numeric values, while the second is a vector containing references to variables that were previously defined. Mixing numeric and reference entries would also be acceptable.

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Vector Variable Definitions:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
<variableDef name="vector1_nd" varID="vector1" units="nd">
  <description>
    An example of encoding a vector of data in a
    DAVE-ML compliant XML file.
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows -->
  </dimensionDef>
  <array>
    <dataTable>1.0, 2.0, 3.0</dataTable>
  </array>
  <isOutput/>
</variableDef>

<variableDef name="eulerAngle_rad" varID="eulerAngles" units="rad">
  <description>
    An example of encoding a vector of references in a
    DAVE-ML compliant XML file.
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows -->
  </dimensionDef>
  <array>
    <dataTable>
      eulerAzimuthAngle_,
      eulerInclinationAngle_,
      eulerRollAngle_
    </dataTable>
  </array>
  <isOutput/>
</variableDef>
```

8.3 Examples of Matrix Variable Definitions

The following are sample matrix variable definitions. The first is a 2 dimensional matrix of numeric values, while the second is a matrix containing references to variables that were previously defined. The second matrix represent the inertia tensor of a body. Mixing numeric and reference entries would also be acceptable.

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      Matrix Variable Definitions:
      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
 <!-- Internal -->
 <variableDef name="identityMatrix" varID="identityMatrix" units="">
   <description>
     An identity matrix.
   </description>
   <dimensionDef>
     <dim>3</dim> <!-- Number of columns -->
     <dim>3</dim> <!-- Number of rows -->
   </dimensionDef>
   <array>
     <dataTable>
       <!-- Row #1 -->
       1.0, 0.0, 0.0,
       <!-- Row #2 -->
       0.0, 1.0, 0.0,
       <!-- Row #3 -->
       0.0, 0.0, 1.0,
     </dataTable>
   </array>
 </variableDef>

 <!-- Output -->
 <variableDef name="inertiaTensor_kg.m2" varID="inertiaTensor" units="kg.m2">
   <description>
     An example of encoding a matrix of references in a
     DAVE-ML compliant XML file.
   </description>
   <dimensionDef>
     <dim>3</dim> <!-- Number of columns -->
     <dim>3</dim> <!-- Number of rows -->
   </dimensionDef>
   <array>
     <dataTable>
       <!-- Row #1  IXX, -IXY, -IXZ -->
       inertiaIXX, -inertiaIXY, -inertiaIXZ,
       <!-- Row #2 -IXY,  IYY, -IYZ -->
       -inertiaIXY, inertiaIYY, -inertiaIYZ,
       <!-- Row #3 -IXZ, -IYZ,  IZZ -->
       -inertiaIXZ, -inertiaIYZ, inertiaIZZ,
     </dataTable>
   </array>
   <isOutput/>
 </variableDef>
```

8.4 Examples of Vector and Matrix Operations

The following are examples using vector and matrix variables in calculations for computing related variables. The examples include:

- computing the magnitude (or norm) of a vector,
- calculating the scalar (dot) and outer products using the transpose MathML syntax,
- calculating the scalar product using the MathML syntax,
- using the inverse MathML syntax with the vector (cross) product to compute a vector, in this case the Euler angle rates,
- calculating the determinant of a square matrix,
- and extraction of various components from a matrix

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Norm Example:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
<variableDef name="normValue" varID="normValue" units="">
  <description>
    This represents the magnitude (or norm) of a vector,
    that is the square root of the sum of squares of the
    individual elements.

    It can be computed as:
      result = square root( dot( V1, V1)), or
             = square root( scalarproduct( V1, V1)) using the MathML tags

    Or alternatively, 'norm' could be defined as a csymbol function (?)

    For vector1, defined previously, the result would be the
    square root of 14, which is approximately equal to 3.74166
  </description>
  <calculation>
    <math>
      <apply>
        <root/>
        <apply>
          <scalarproduct/>
          <ci>vector1</ci>
          <ci>vector1</ci>
        </apply>
        <ci>Alpha</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Transpose Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
<variableDef name="transposeToScalar" varID="transposeToScalar" units="">
  <description>
    This represents an example of the transpose directive to form a scalar.
       vector1: [3,1]; eulerAnlges: [3,1]
       ~vector1: [1,3]; eulerAnlges: [3,1]

       result = ~vector1 * eulerAnlges: [1,1] (scalar)

    This example is equivalent to the scalar (dot) product of two vectors.
  </description>
  <calculation>
    <math>
      <apply>
        <times/>
        <apply>
          <transpose/>
          <ci>vector1</ci>
        </apply>
        <ci>eulerAngles</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>

<variableDef name="transposeToMatrix" varID="transposeToMatrix" units="">
  <description>
    This represents an example of the transpose directive to form a matrix.
    vector1: [3,1]; eulerAnlges: [3,1]
    vector1: [3,1]; ~eulerAnlges: [1,3]

    result = vector1 * ~eulerAnlges: [3,3]

    This example is equivalent to the 'outerproduct' of two vectors.
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of columns -->
    <dim>3</dim> <!-- Number of rows -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <times/>
        <ci>vector1</ci>
        <apply>
          <transpose/>
          <ci>eulerAngles</ci>
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Inverse & Vector (Cross) product Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
<variableDef name="eulerRates_rad_s" varID="eulerRates" units="rad_s">
  <description>
    This represents the calculation of the Euler rates from known
    angular velocities, the inertia tensor and the applied moments.

    result = !inertiaTensor * (( -angularVelocity X
                                  ( inertiaTensor * angularVelocity)) +
                                moment)

    MathML defines the cross product as the 'vectorproduct', which is the
    tag used in this example.
  </description>
  <dimensionDef>
    <dim>3</dim> <!-- Number of rows -->
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <times/>
        <apply>
          <inverse/>
          <ci>inertiaTensor</ci>
        </apply>
        <apply>
          <plus/>
          <apply>
            <vectorproduct/>
            <ci>-angularVelocity</ci>
            <apply>
              <times/>
              <ci>inertiaTensor</ci>
              <ci>angularVelocity</ci>
            </apply>
          </apply>
          <ci>moment</ci>
        </apply>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Scalar (Dot) Product Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
<variableDef name="dotProduct" varID="dotProduct" units="">
  <description>
    This represents the calculation of the dot product of two vectors.
    The result is a scalar.

    MathML defines the dot product as the 'scalarproduct', which is the
    tag used in this example.
  </description>
  <calculation>
    <math>
      <apply>
        <scalarproduct/>
        <ci>vector1</ci>
        <ci>eulerAngles</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>


<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Determinant Examples:
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->
<variableDef name="determinantInertia" varID="determinantInertia" units="">
  <description>
    This represents the calculation of the determinate of a matrix.
    The matrix must be square to compute the determinant. The result
    is a scalar.
  </description>
  <calculation>
    <math>
      <apply>
        <determinant/>
        <ci>inertiaTensor</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

```
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Selector Examples:

     element,
     row,
     column,
     diag, planar or co-planar (?)
     mslice

     MathML has a 'selector' tag, which is of limited capability. It
     is defined for lists, vectors and 2-D Matrices, and as documented,
     its functionality permits the extractions of individual entries
     and rows from matrices. It does not appear to be suitable for
     extracting matrix columns, diagonals or sub-matrices, or working
     with matrices of higher than 2 dimensions.

     An extension to the 'selector' tag could be developed. The examples
     shown here use the 'other' attribute to provide context for the
     'selector' tag having a entries of either:
        element: for the extraction of an entry,
        row    : for the extraction of a row from a matrix,
        column : for the extractions of a column from a matrix,
        diag   : for the extraction of a diagonal vector from a matrix, and
        mslice : for the extraction of a sub-matrix.

     The number of arguments that need to be set depends on the selection
     of the 'other' attribute and the dimension of the vector or matrix.

     The argument immediately after the 'selector' tag is the name of
     the vector or matrix from which data is to be extracted. The following
     arguments define the plane, row or column to be extracted, or where
     extraction starts.

     The size of the argument list for the 'selector' tag is defined as
     nary - or effectively undefined. However, in the MathML documentation
     it states that a maximum of three arguments would be expected - which
     includes specification of the vector or matrix. The limitation of
     a maximum of 3 arguments is removed for its use as presented in
     these examples, where the maximum number of arguments is instead
     related to the dimension of the vector or matrix being processed.

     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  -->

<variableDef name="elementSelection" varID="elementSelection" units="">
  <description>
    This represents the process for extracting an entry from a vector
    or matrix. The result is a scalar.

    The element extracted is IYY.
    MathML numbers rows and columns of vectors and matrices from 1.
    Thus IYY is element [2,2]
  </description>
  <calculation>
    <math>
      <apply>
        <selector other="element"/>
        <ci>inertiaTensor</ci>
```

```
        <!-- Need one argument for each dimension of the vector or matrix -->
        <cn>2</cn> <!-- Row number of element -->
        <cn>2</cn> <!-- Column number of element -->
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>

<variableDef name="rowSelection" varID="rowSelection" units="">
  <description>
    This represents the process for extracting a row from a matrix.
    The result is a vector.

    The row extracted is IXZ, IYZ, IZZ.
    MathML numbers rows and columns of vectors and matrices from 1.
    Thus the row is #3
  </description>
  <dimensionDef>
    <!-- Number of entries in the resulting vector, #columns of matrix -->
    <dim>3</dim>
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="row"/>
        <ci>inertiaTensor</ci>
        <!-- Need arguments selecting plane of matrix when 3-D or more -->
        <!-- Need one argument for the row of the matrix -->
        <cn>3</cn> <!-- Row number to extract -->
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>

<variableDef name="columnSelection" varID="columnSelection" units="">
  <description>
    This represents the process for extracting a column from a matrix.
    The result is a vector.

    The column extracted is IXY, IYY, IZY.
    MathML numbers rows and columns of vectors and matrices from 1.
    Thus the column is #2
  </description>
  <dimensionDef>
    <!-- Number of entries in the resulting vector, #rows of matrix -->
    <dim>3</dim>
  </dimensionDef>
  <calculation>
    <math>
      <apply>
        <selector other="column"/>
        <ci>inertiaTensor</ci>
        <!-- Need argument s selecting plane of matrix when 3-D or more -->
        <!-- Need one argument for the column of the matrix -->
        <cn>2</cn> <!-- Column number to extract -->
```

```
        </apply>
      </math>
    </calculation>
    <isOutput/>
  </variableDef>

  <variableDef name="diagSelection" varID="diagSelection" units="">
    <description>
      This represents the process for extracting a planar diagonal
      from a matrix.
      The result is a vector.

      The diagonal extracted is IXX, IYY, IZZ.
      MathML numbers rows and columns of vectors and matrices from 1.
      The starting element in the matrix is [1,1]
    </description>
    <dimensionDef>
      <dim>3</dim> <!-- The number of entries in the resulting vector -->
    </dimensionDef>
    <calculation>
      <math>
        <apply>
          <selector other="diag"/>
          <ci>inertiaTensor</ci>
          <!-- Need argument selecting plane of matrix when 3-D or more -->
          <!-- Need one argument for the starting row number of the matrix -->
          <!-- Need one argument for the starting column number of the matrix
-->
          <cn>1</cn> <!-- Row number to start extraction -->
          <cn>1</cn> <!-- Column number to start extraction -->
        </apply>
      </math>
    </calculation>
    <isOutput/>
  </variableDef>

  <variableDef name="matrixSlice" varID="matrixSlice" units="">
    <description>
      This represents the process for extracting a matrix from a matrix.
      The result is a matrix.

      The dimension of the extracted matrix is equal to or less than the
      dimension of the original matrix. The number of entries extracted
      in each dimension is defined by the dimensionDef properties for
      the sub-matrix.

      The element from which to start extracting the sub-matrix needs to
      defined.

      The matrix extracted is IXX, IXY, IYX, IYY.
      MathML numbers rows and columns of vectors and matrices from 1.
      The starting element in the matrix is [1,1],
    </description>
    <dimensionDef>
      <dim>2</dim> <!-- The number of rows in sub-matrix -->
      <dim>2</dim> <!-- The number of columns in sub-matrix -->
    </dimensionDef>
```

```
    <calculation>
      <math>
        <apply>
          <selector other="mslice"/>
          <ci>inertiaTensor</ci>
          <!-- Need one argument for the starting row of the matrix -->
          <!-- Need one argument for the starting column of the matrix -->
          <cn>1</cn> <!-- Row number to start extraction -->
          <cn>1</cn> <!-- Column number to start extraction -->
        </apply>
      </math>
    </calculation>
    <isOutput/>
  </variableDef>
```

9 References

[BSR/AIAA S-119] Flight Dynamic Model Exchange Standard.

[DAVE-ML] Bruce Jackson, *Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) Reference Version 2.0 RC 3.0.*

[Hubbard] John H. Hubbard, Barbara Burke Hubbard, *Vector Calculus, Linear Algebra and Differential Forms – A Unified Approach*, Prentice Hall, New Jersey, 1999.

[Kuo] Benjamin C. Kuo, Automatic Control Systems 5th Ed, Prentice Hall, New Jersey, 1987.

[MathML] Mathematical Markup Language (MathML) Version 2.0 (Second Edition), W3C, http://www.w3.org/TR/MathML2/, 2003.