**Proposed MOSDEX Syntax for Nonlinear Problems**

Dr. Jeremy A. Bloom  
[jeremyblmca@gmail.com](mailto:jeremyblmca@gmail.com)  
July 13, 2019

1. **Overview**

This paper proposes extensions to MOSDEX to enable specification of nonlinear problems. These extensions are based on the JSON ADFun proposal by Brad Bell at <https://coin-or.github.io/CppAD/doc/json_ad_graph.htm> and <https://coin-or.github.io/CppAD/doc/to_json.cpp.htm>.

The primary extension to MOSDEX is a new class called Expression that contains a list of Nodes that comprise an expression graph, analogously to the ADFun graph construct created by Brad. Nodes themselves are like Tables (i.e. they each have a defined schema, and they can have Instance or Recipe form). The main difference between MOSDEX and JSON ADFun is that in the latter, a node is essentially a scaler object while in MOSDEX, a node is a kind of Table that has dimensionality as specified by its key fields. Thus MOSDEX implements families of Expressions that correspond to the structure of the optimization problem that it represents.

In order to explain the proposal, two examples have been created at <https://github.com/coin-modeling-dev/MOSDEX-Examples>. Both are representations of a quadratically constrained network flow problem presented by Fourer at <https://ampl.com/MEETINGS/TALKS/2015_11_Philadelphia_MC19.pdf>. The trafficNetworkQP <https://github.com/coin-modeling-dev/MOSDEX-Examples/blob/master/trafficNetworkQPNew.json> shows the MOSDEX representation of this quadratic program. The trafficNetworkNLP <https://github.com/coin-modeling-dev/MOSDEX-Examples/blob/master/trafficNetworkNLPNew.json> shows the same problem using the NLP extension of MOSDEX.

Note that the proposed extension is tentative, and readers are invited to comment and to suggest alternatives.

1. **The Expression and Node Classes**

Expression is a new top-level class in MOSDEX; as named JSON Objects, multiple Expression objects can coexist within a MOSDEX Problem, typically one for each nonlinear expression in the Problem. Within an Expression object, a list (i.e. JSON Array) of Nodes represents the subexpressions comprising the expression. Each node is one of 4 types: Independent Variable, Dependent Variable, Parameter, or Operation. Independent and Dependent Variables link to a Variable modeling object; the distinction between them is that Independent Variables are inputs to the expression and Dependent Variables are outputs. Parameters link to data items and do not interact with the solver. Finally, Operations represent operators that combine Independent Variables and Parameters; typically they represent elementary math operations such as add or multiply, or math functions, such as exp or log. Currently MOSDEX offers unary (one operand) and binary (two operands) operators, but a future extension will also permit operators with an indeterminate number of operands (e.g. sum) which are needed to support model/data separation. Operations combine in a tree-like fashion that enables building complex expressions from such elementary operations. The use of expression trees is common to representing nonlinear optimization problems, and it facilitates automatic differentiation used by nonlinear solvers. The nodes of an expression should take the form of a tree (each parent can have multiple children but each child has only one parent); however, MOSDEX will not check this condition; that is the responsibility of the service that evaluates the expression when called by the solver.

The Nodes of an Expression are held in a list (JSON Array) so that they don’t need to be named. Instead, each member of the Nodes list has a numerical index and is referenced by that index (e.g. Nodes[0] is the first element of the Nodes list). Each schema of each node has fields for the index (ID), Operation, its keys, its Operands, and its Result. The result (e.g. Nodes[0].Result) is usually used as an operand in the parent node. A node object also must contain either an Instance or Recipe specifying its connections to other Tables in the Problem. In the case of an Independent Variable, the link is often through a call to the Solver which retrieves the current value of the variable (assuming that the Solver repeatedly needs to evaluate the expression for different trial solutions). A Dependent Variable links back to a Constraint or Objective through a Term object.

1. **Additional Modifications to MOSDEX**

In order to implement expressions, several modifications of the existing MOSDEX syntax are proposed. First, the Coefficient class is renamed Term, and it needs to include a Type field to represent terms in linear, quadratic, or nonlinear expressions. Second, the Expression field in the Heading object is renamed Math to avoid ambiguity.