DALE W. JORGENSON ASSOCIATES

1010 MEMORIAL DRIVE, 14C

CAMBRIDGE, MASSACHUSETTS 02138

Technical Work Plan

Project: Construction of Open Source Model of US Economy -- IGEM

1 Background and purpose

The Science Advisory Board (SAB) to the US Environmental Protection Agency (EPA) in its report, *SAB Advice on the Use of Economy-Wide Models in Evaluating the Social Costs, Benefits, and Economic Impacts of Air Regulations* submitted to the EPA Administrator in September 2017, noted that "CGE (*sic,* computable general equilibrium) models could provide several important benefits as a supplement to the EPA's current set of analytical tools for the analysis of air regulations." However, the SAB also noted that CGE modeling is challenging due to their complexity, large scale. formidable data requirements and a lack of transparency ("the mechanisms driving their results can be opaque"). The SAB thus recommends that the EPA "initiate and support a third-party open-source program to assemble and maintain publicly available high-quality datasets." It says that "such a database is an example of a public service that … will benefit public and private decision makers". The SAB notes that the EPA has been a leader in the development of benefit-cost analysis, and recommends that the agency "continue that leadership by beginning to integrate economy-wide modeling, and computable general equilibrium modeling in particular, into regulatory analysis in order to offer a more comprehensive assessment of the benefits and costs of air regulations."

In line with the recommendations of SAB (2017) for EPA to support an open-source database suitable for CGE models and to integrate CGE models into regulatory analysis, Dale W. Jorgenson Associates (DJA) proposes to develop an open-source version of their Intertemporal General Equilibrium Model (IGEM) that will be available and usable by EPA staff and the interested public. This Open-IGEM will be a self-contained package of data and model code that someone familiar with CGE models will be able to use. In addition to its accessibility, Open-IGEM will be complete and transparent in its data sourcing, model documentation and guides to use and application.

1.1 Overview of IGEM and its use

IGEM reflects a more than four-decade evolution in economic theory, data development, econometric methodology, model construction, algorithmic solution and application and use. It is based on solid microeconomic foundations of producer and consumer behaviors with estimated parameters that conform to their theoretical properties. IGEM remains unique in its application of the translog flexible functional form with parameters estimated from observed market behavior revealed over forty to fifty years. Every aspect of this evolution has appeared in peer-reviewed publications and forums, including those sponsored by the EPA and its SAB and several involving multi-model comparisons (e.g., the studies of Stanford University's Energy Modeling Forum (EMF)).

IGEM is a neo-classical model of economic structure and growth. It is based on a unified accounting framework consistent with the *US National Income and Product Accounts* and selected industry classification systems. IGEM covers all aspects of long-run growth – the supply of capital, labor, imported and intermediate inputs to production, the rates and directions of technical change (both autonomous and price-induced) for each producing sector, and the degrees of substitutability among inputs and commodities in production and final demand (consumption, investment, governments and foreign trade).

Within IGEM, there are the following substitution possibilities:

- Among the inputs to production – capital, labor, energy and materials and the details of energy and materials;
- In household decisions
    - Saving or the intertemporal choice between present versus future full consumption (i.e., consumer goods, capital and consumer services and leisure);
    - Among consumer goods, capital and consumer services and leisure for households categorized by family size, race and sex of the household head, rural-urban status, and region, the leisure choice determining labor supply;
    - Among consumer goods and services;
- Among types of investment and capital goods;
- Between competing domestic and imported goods and services;
- Among exported supplies of goods and services;
- Among the components of final demand – household consumption, private investment, government purchases and exports.

In IGEM, capital accumulation arises from the saving and investment behavior of households and businesses and provides this essential primary input to production and consumption. IGEM contains both backward- and forward-looking dynamics as capital availability results from previous investment and capital goods prices reflect the discounted present value of future capital service flows. Household decisions occur with perfect foresight on prices, interest rates and full consumption. Capital and labor are mobile across all producing and consuming sectors.

In solution, IGEM ensures market balances (supply equals demand) in value and quantity terms, including limits placed on private investment dictated by domestic and foreign saving behavior and by the fiscal policies of federal, state, and local governments. Solutions include traditional measures of economic performance (e.g., real GDP, commodity prices and quantities) as well as those that focus on individual, household and societal welfare, the latter in terms of both efficiency and equity.

Projections depend on extrapolations of historical trends and calibrations to key driving variables and input assumptions (e.g., population, world oil prices, budget deficits and trade balances, domestic tax policies, etc.). Model outcomes portray "best case" scenarios conditional on the adjustment costs implicit in IGEM's econometrically estimated parameters.

As part of IGEM's history of applications for EPA, in 2001-2002, DJA conducted economic analyses of the benefits and costs of the Clean Air Act, 1970-1990, and the Clean Air Act Amendments, 1990-2010, for EPA's National Center for Environmental Economics. For EPA's Office of Atmospheric Programs, DJA conducted economic analyses of the following Congressional climate policy initiatives:

- American Power Act of 2010 (June 2010)
- American Clean Energy and Security Act of 2009 (June 2009)

- Waxman-Markey Discussion Draft of the American Clean Energy and Security Act of 2009 (April 2009)
- Lieberman-Warner Climate Security Act of 2008 (March 2008)
- Low Carbon Economy Act of 2007 (September 2007 and January 2008)
- Climate Stewardship and Innovation Act of 2007 (July 2007)

The newest version of IGEM is based on the North American Industry Classification System (NAICS) and is estimated econometrically over a time series of IO tables covering the years 1960-2010. It is structured around 36 industries and commodities – 1 each of agriculture, construction, and transportation, 6 of energy, 15 of mining and manufacturing and 12 of services. Oil and gas mining now are separated and along with coal, refined petroleum and electric and gas utilities comprise IGEM's energy sectors. The manufacturing and services groups offer a more contemporary view of the U.S. economy; information technology sectors – hardware and software, wholesale and retail trade, finance, business services, education, and health and welfare are among the revised model aggregates. The inaugural application of this version of IGEM is EMF's multi-model analysis of carbon taxation and revenue recycling (EMF 32).

1.2 Design criteria for Open IGEM

The overarching goal of the project is to produce an open source version of IGEM that can be widely used by researchers and policy analysts. To achieve that, the model's implementation must have the following key characteristics:

- *Transparency*. The model should be expressed in a form that allows users to focus on the economic relationships that define it rather than the computational overhead needed to solve it. Doing so makes the model's underlying mechanisms as clear as possible to users, or to outside parties evaluating an analysis carried out with the model. In addition, the model's solution procedure should be well-established and clearly documented.

- *Flexibility*. The implementation should allow users to adapt the model to new uses, or to extend or revise the model to incorporate new data or improved theoretical models of behavior. Ideally, the open version of the model should be a platform for facilitating a wide range of new research and analysis. In addition, the implementation should be flexible in the range of simulations it allows by providing flexible partitioning of the model's variables into endogenous and exogenous sets. For example, it should be possible to run a simulation endogenously calculating the tax rate (usually exogenous) needed to achieve a particular revenue target (usually endogenous).

- *Reliability*. The implementation should reduce or eliminate many common kinds of coding errors that can lead to models that are incorrect in subtle ways that are difficult to detect and debug. This aspect is particularly important since the goal of the project is to enable new users with less experience in large scale modeling to carry out high-quality analysis. In addition, the model should solve reliably: the solution procedure should not fail to find a solution when one exists (i.e., it should not fail for correctly specified models), nor should it require sophisticated tuning and adjustment on the part of the user.

Finally, intertemporal models with forward-looking agents and saddle-path stability should solve reliably.

- *Capacity*. The implementation should be capable of handling models with a high degree of dimensionality, including large numbers of sectors and households. In addition, it should support a high degree of intertemporal detail, allowing models to include large numbers of state and costate variables (e.g., models with sector-specific capital), and allowing transversality conditions at the model's terminal steady state or balanced growth equilibrium to be applied in a straightforward manner.

2 The Structure of Open IGEM

2.1 Introduction to Open IGEM

To achieve these goals, we have designed Open IGEM to consist of two parts to separate the building of the economics of the model from the building of the executable code.

- Front-end part contains the equations of the model in a high-level form that reads almost like the algebraic form as written in the Model Equation Appendix. It also contains the data structures in a high-level form.
- Back-end part contains the model in GEMPACK form; this consists of the equations written in GEMPACK code and data files that are readable by GEMPACK so that the model can be solved.

With this separation of the model's economics from the solution algorithms it uses, we allow the typical user to focus on the theory of the economic model without worrying about the complexity of learning the GEMPACK system and coding it. This structure also allows the user to use a different model solution software package other than the GEMPACK system used here, for example, GAMS. Implementing with a different software system would not be trivial, but would be much easier given the front-end piece. The remainder of this Section 2 describes the front-end piece and Section 3 describes the implementation in GEMPACK.

The main feature of the front-end part is the model equations written in a portable, general-purpose language called Sym developed by McKibbin and Wilcoxen (2013). Sym is developed to facilitate the translation of model equations written in algebraic form to the form used by the solution software such as GEMPACK or GAMS. Sym is a set-driven matrix language that descends from GAMS and GEMPACK. It imposes rigorous conformability rules on all expressions to eliminate a broad range of potential errors in the design and coding of the model. A useful consequence of these rules is that subscripts are generally unnecessary and the model can be expressed very concisely and cleanly. Sym is described in more detail in section 2.3 below.

2.2 Overview of key components of Open IGEM

The distribution package of Open IGEM will consist of a large set of files. Here we summarize the components of this intended system.

- An equation appendix in a Word file listing all the model equations in algebraic form. This consist of two parts, the first is the set of equations defining the intertemporal equilibrium solution of the model. The second part consists of post-solution equations giving variables of interest that depends on the endogenous variables in the first part, e.g. energy flows in BTUs, welfare indices, productivity decompositions, etc.
- A set of files of the model coded in the Sym language. These files define the sets, variable names and equations.
- The GEMPACK ready model file generated by Sym; i.e. the file with the information to set up the GEMPACK sets and the equations written in the GEMPACK TABLO form.
- One or more parameter files in CSV form (or another widely used format); this is a list of parameter names and their values. Parameter values do not change over time.
- File with list of variables, and indicators as to which are (typically) exogenous and which endogenous. A short description of the variable goes with each one.
- Files in CSV form with data on baseline values of exogenous variables, i.e. data for each period in the solution horizon. These include population variables, world commodity prices, government deficits, fiscal parameters such as tax rates and depreciation schedules, exogenous portions of productivity change (latent terms in the cost functions), the productivity terms may include those that are used to calibrate, say, the growth rate of GDP if the model user requires the solution to meet a particular growth path.
- Files with the baseline solution; i.e. values of the endogenous variables for each period produced by the solution software (GEMPACK). The set of endogenous variables is large and tools will be provided to enable users to choose which ones to store in the solution files.
- A set of files with an example of a policy simulation exercise.

2.3 Computational form of the model's equations

Open IGEM's equations would be provided as a set of files written in Sym, a high-level modeling language developed by McKibbin and Wilcoxen (2013). Sym is a set-driven language that descends from GAMS and GEMPACK. It has several features that make it attractive for use with Open IGEM. First, it handles all subscripting automatically and allows models with high dimensionality to be expressed in a clean and concise form very close to matrix notation. Second, models expressed in Sym can be translated automatically into forms suitable for a range of backend solution packages. As a result, it cleanly separates the economic definition of the model from the solution algorithm, and it allows the model to be moved from one package to another very easily if needed. Complete code for the translation program will be made open source as part of the project.

Most importantly, however, Sym imposes rigorous conformability rules on each operation within each equation, which eliminates a broad range of potential errors in the design and coding of a model. Sym's conformability rules are designed to prevent an important problem that arises in large-scale computational models: incorrectly-coded equations that are mathematically legitimate but economically nonsensical. Detecting and debugging such equations can be quite difficult.

For example, suppose a model has four households that represent US regions from the northeast to the southwest (ne, se, nw, sw) and each household buys four inputs that represent capital, labor, energy and materials (k, l, e, m). Total spending by each household (v) could be computed from household purchases of each good (q) and the corresponding good's price (p) as shown in (1):

$$(1) \qquad \begin{bmatrix} v_{ne} \\ v_{se} \\ v_{nw} \\ v_{sw} \end{bmatrix} = \begin{bmatrix} q_{ne,k} & q_{ne,l} & q_{ne,e} & q_{ne,m} \\ q_{se,k} & q_{se,l} & q_{se,e} & q_{se,m} \\ q_{nw,k} & q_{nw,l} & q_{nw,e} & q_{nw,m} \\ q_{sw,k} & q_{sw,l} & q_{sw,e} & q_{sw,m} \end{bmatrix} \begin{bmatrix} p_k \\ p_l \\ p_e \\ p_m \end{bmatrix}$$

However, small coding errors can easily lead incorrect implementations. For example, in (2), the transpose of q has been used:

$$(2) \qquad \begin{bmatrix} v_{ne} \\ v_{se} \\ v_{nw} \\ v_{sw} \end{bmatrix} = \begin{bmatrix} q_{ne,k} & q_{se,k} & q_{nw,k} & q_{sw,k} \\ q_{ne,l} & q_{se,l} & q_{nw,l} & q_{sw,l} \\ q_{ne,e} & q_{se,e} & q_{nw,e} & q_{sw,e} \\ q_{ne,m} & q_{se,m} & q_{nw,m} & q_{sw,m} \end{bmatrix} \begin{bmatrix} p_k \\ p_l \\ p_e \\ p_m \end{bmatrix}$$

Another possible error is shown in (3), where the order of elements in vector p has been permuted and no longer aligns with the columns of q:

$$(3) \qquad \begin{bmatrix} v_{ne} \\ v_{se} \\ v_{nw} \\ v_{sw} \end{bmatrix} = \begin{bmatrix} q_{ne,k} & q_{ne,l} & q_{ne,e} & q_{ne,m} \\ q_{se,k} & q_{se,l} & q_{se,e} & q_{se,m} \\ q_{nw,k} & q_{nw,l} & q_{nw,e} & q_{nw,m} \\ q_{sw,k} & q_{sw,l} & q_{sw,e} & q_{sw,m} \end{bmatrix} \begin{bmatrix} p_e \\ p_k \\ p_l \\ p_m \end{bmatrix}$$

A third error, as shown in (4), would be to use a price defined on a completely different domain, such as goods a to d rather than inputs k to m:

$$(4) \qquad \begin{bmatrix} v_{ne} \\ v_{se} \\ v_{nw} \\ v_{sw} \end{bmatrix} = \begin{bmatrix} q_{ne,k} & q_{ne,l} & q_{ne,e} & q_{ne,m} \\ q_{se,k} & q_{se,l} & q_{se,e} & q_{se,m} \\ q_{nw,k} & q_{nw,l} & q_{nw,e} & q_{nw,m} \\ q_{sw,k} & q_{sw,l} & q_{sw,e} & q_{sw,m} \end{bmatrix} \begin{bmatrix} p_a \\ p_b \\ p_c \\ p_d \end{bmatrix}$$

Under standard linear algebra, which only requires conformability between the number of columns of q and the number of rows of p, (2) through (4) are all legitimate and can be computed: none will be flagged as incorrect by conventional software. That makes them hard to detect and correct: such a model will usually run, but it will produce invalid results that may not be obviously wrong.

Languages such as GEMPACK avoid many of problems noted above by foregoing matrix notation and requiring explicit subscripting of all variables in all equations. Sym takes a

different approach. Rather than requiring equations to be written in scalar notation, it imposes a set of conformability rules that allow it to deduce the appropriate alignment of variables unambiguously in each operation. In effect, the rules allow it to handle all subscripting automatically. The result is that Sym preserves the clarity and simplicity of matrix notation without introducing the opportunities for error discussed above.

Example application:

To illustrate how Sym works we now consider how the calculation above would be implemented. First, Sym files consist of three types of statements: set declarations, parameter and variable declarations, and equations. All statements end with a semicolon, and any text after two consecutive slashes (//) and continuing to the end of the line is taken to be a comment and is ignored. Figure 1 shows an example of Sym code that implements a few calculations involving the household expenditures mentioned above. It begins with declarations that create a set of time periods (discussed in detail in a subsequent section), a set of households (ne, se, nw, sw), and a set of inputs (k, l, e, m). The next block of statements defines the variables: the price of each input (p); the quantity of each input purchased by each household (q); the value of spending on each input by each household (v); the total value of spending by each household (vby); and the total value of spending on each input (von).

Figure 1: Example of Sym Code

```
set time (t0,t1,t2,t3,t4,t5) ;
set households (ne,se,nw,sw) ;
set inputs (k,l,e,m) ;

variable p(inputs) ;
variable q(households, inputs) ;
variable v(households, inputs) ;
variable vby(households) ;
variable von(inputs) ;

v = q*p ;
vby = sum(inputs, v) ;
von = sum(households, v) ;
```

The final lines in Figure 1 are equations defining the three value variables. In the first equation, Sym interprets $v = q * p$ to mean an element-by-element operation: each element of v will be the product of an appropriate element of q multiplied by an appropriate element of p. It then applies a set of conformability rules to ensure that elements of the three variables (v, q and p) can be aligned unambiguously. In this case, two conformability rules apply. The first applies all multiplication operations, and in this case it requires that the set over which p is defined must exactly match one of the sets over which q is defined. The second rule applies to all equality operations, and in this case it requires that v and $q * p$ must be defined over identical sets.

In the example, the multiplication rule is satisfied because p and q are both defined over $inputs$. This allows Sym to determine unambiguously which price goes with each quantity: the elements of p and q are matched according to the set they have in common, $inputs$. As a result, $q * p$

will have elements such as $q_{ne,k}p_k$ and $q_{sw,m}p_m$, as intended. Because Sym uses the common set to align q and p, it prevents the error shown in (2), where q was transposed. As a result, unlike under standard matrix notation, multiplication under Sym is commutative: $q * p$ and $p * q$ will produce identical results. Finally, because Sym aligns variables based on the names of the elements in each set, not on the order in which they were listed in the set declaration, the calculation is invariant to permutations in the list of elements. This eliminates the error shown in (3), where the order of elements in p was permuted.

The rule also allows Sym to catch the error in (4) where p was defined over an incorrect set (that is, it was a vector of prices for the wrong set of things). For example, the code in Figure 2 defines p over the set $goods$ rather than the set $inputs$ (differences from Figure 1 are shown in bold). Since p and q no longer have a set in common, Sym generates the compile-time error message shown in Figure 3 indicating that $q * p$ is not conformable.

Figure 2: Sym Code Triggering a Conformability Error

```
set time (t0,t1,t2,t3,t4,t5) ;
set households (ne,se,nw,sw) ;
set inputs (k,l,e,m) ;
set goods (a,b,c,d) ;

variable p(goods) ;
variable q(households, inputs) ;
variable v(households, inputs) ;
variable v_by(households) ;
variable v_on(inputs) ;

v = q*p ;
vby = sum(inputs, v) ;
von = sum(households, v) ;
```

Figure 3: Conformability Error Message

```
Equation 1
----------

Input File Error:
   Arguments not Conformable
      Expression: q*p
      Left side:  q
      Domain:     households,inputs,time
      Right side: p
      Domain:     goods,time
```

The equality rule is simpler but it still eliminates potential errors. It ensures that there is exactly one unambiguous element of v for each element of $q * p$. Together, the two rules produce the set of calculations shown in (5):

$$(5) \qquad \begin{bmatrix} v_{ne,k} & v_{ne,l} & v_{ne,e} & v_{ne,m} \\ v_{se,k} & v_{se,l} & v_{se,e} & v_{se,m} \\ v_{nw,k} & v_{nw,l} & v_{nw,e} & v_{nw,m} \\ v_{sw,k} & v_{sw,l} & v_{sw,e} & v_{sw,m} \end{bmatrix} = \begin{bmatrix} q_{ne,k}p_k & q_{ne,l}p_l & q_{ne,e}p_e & q_{ne,m}p_m \\ q_{se,k}p_k & q_{se,l}p_l & q_{se,e}p_e & q_{se,m}p_m \\ q_{nw,k}p_k & q_{nw,l}p_l & q_{nw,e}p_e & q_{nw,m}p_m \\ q_{sw,k}p_k & q_{sw,l}p_l & q_{sw,e}p_e & q_{sw,m}p_m \end{bmatrix}$$

As indicated in equation (5), in Sym the expression $q * p$ does not imply summing along either of the dimensions (as would be the case with standard matrix notation). Instead, as shown in the final two lines of Figure 1, the sum is taken explicitly via a *sum* operator that indicates the dimension over which the sum is to be computed. As a result, the two equations are clean and easy to interpret because they closely match the logic of their conventional scalar counterparts, shown below, except that subscripts are implicit:

$$(6) \qquad vby_h = \sum_{g\epsilon\{cap,\dots,mat\}} p_g \cdot q_{h,g}\,, \ \forall h\epsilon\{ne,\dots,sw\}$$

$$(7) \qquad von_g = \sum_{h\epsilon\{ne,\dots,sw\}} p_g \cdot q_{h,g}\,, \ \forall g\epsilon\{cap,\dots,mat\}$$

There is no need for the equation to include explicit subscripts because all subscripting is handled internally by Sym.

Sym's conformability rules apply to arrays with more dimensions as well. For example, suppose the calculation above were to be implemented in a model with two regions, the US (us) and the rest of the world (row). In that case, as shown in Figure 4, the variables would each be defined over the additional set (differences from Figure 1 shown in bold). However, no changes would be needed in the three equations because their meaning can be determined unambiguously from the definitions of the variables.

Figure 4: Example Extended to Multiple Regions

```
set time (t0,t1,t2,t3,t4,t5) ;
set households (ne,se,nw,sw) ;
set inputs (k,l,e,m) ;
set regions (us,row) ;

variable p(inputs, regions) ;
variable q(households, inputs, regions) ;
variable v(households, inputs, regions) ;
variable v_by(households, regions) ;
variable v_on(inputs, regions) ;

v = q*p ;
vby = sum(inputs, v) ;
von = sum(households, v) ;
```

The multiplication operation would again be aligned on the common sets between q and p, in this case $inputs$ and $regions$, and the equality operation would again be aligned on identical elements in v and $q * p$. For example, purchases of k by the ne household in the US would be computed as shown in (8):

(8) $$v_{ne,k,us} = q_{ne,k,us} p_{k,us}$$

The two sums generalize in the expected way as well: each sums along the indicated dimension of $q * p$:

(9) $$vby_{h,r} = \sum_{g \epsilon \{cap,\ldots,mat\}} p_{g,r} \cdot q_{h,g,r}\,,\ \forall h \epsilon \{ne, \ldots, sw\}$$

(10) $$von_{g,r} = \sum_{h \epsilon \{ne,\ldots,sw\}} p_{g,r} \cdot q_{h,g,r}\,,\ \forall g \epsilon \{cap, \ldots, mat\}$$

As a result, Sym's conformability rules allows the size and scope of the model to be changed simply by changing the sets over which variables are defined. In most cases, the equations need no modifications.

Backend solution packages supported:

As noted earlier, Sym is a language for expressing models clearly and concisely, and for checking the internal logic of the equations. Models written in Sym can be translated automatically via a program, also called Sym, into the native code used by a range of backend solution packages including the matrix programming language Ox (used by the G-Cubed model in McKibbin and Wilcoxen, 2013), GEMPACK, and TROLL code. Extensions to other languages, such as GAMS, are straightforward as the output code generator is cleanly separated from the parser that reads the input files and carries out compatibility checks.

Flexible input format:

Sym allows model code to be broken up into an arbitrary number of files which are then merged when the model is compiled into its backend form. In addition, Sym is order-independent, and the definitions of parameters, variables and equations can be freely mixed in the files. Finally, Sym allows white space (including carriage returns) to be used freely throughout its input.

Features for handling of time periods:

All variables are implicitly defined over an ordered set $time$ that can contain an arbitrary number of periods. Parameters, in contrast, are taken to be invariant and not implicitly defined over the time set. (If time-varying parameters are needed they can be declared as variables rather than parameters). Because $time$ is ordered, the operators $lead()$ and $lag()$ can be used in equations. Figure 5 shows how a typical capital accumulation equation might be implemented in a model with three sectors, each having its own sector-specific investment good and capital stock, that is to be simulated over six time periods ($t0$ to $t5$). Because all variables are implicitly time-dimensioned, the set $time$ is not mentioned in the declarations of the variables inv and cap.

Figure 5: Examples of Time Subscripting

```
set time (t0,t1,t2,t3,t4,t5) ;
set sectors (s1,s2,s3) ;

variable inv(sectors) ;
variable cap(sectors) ;
parameter dep ;

lead(cap) = cap*(1-dep) + inv ;
```

The accumulation equation is equivalent to (11):

(11) $cap_{s,t+1} = cap_{s,t} * (1 - dep) + inv_{s,t}$

It holds for all s in $sectors$ and all t in $time$ other than $t5$: Sym automatically recognizes that the equation can't apply in the last period. That leaves one fewer equation for each capital stock than there are periods of time, which is usually exactly what is desired: the capital stock in period $t0$ would usually be set exogenously.

Also note that this example illustrates how scalars are handled by Sym. Because the depreciation rate, dep, is declared in Figure 5 as a scalar (i.e., the depreciation rate is identical across sectors), it is automatically conformable with higher-dimensioned variables.

For convenience, Sym also implicitly provides singleton sets called $first$ and $last$ that represent the first and last time periods. The $last$ set is convenient for defining equations and variables that apply only at the steady state, which would usually be imposed in the final period of the simulation. It allows the number of time periods to be changed (i.e., by adding $t6$, $t7$, etc.) without requiring the model's equations to be revised to match the change in the final period.

Support for subsets:

Sym includes extensive support for subsets. Subsets can be created in several ways, such as the following which creates a subset pri (for primary factors) by removing two elements from the earlier set $inputs$:

(12) $set\ pri = inputs - (e, m);$

Figure 6 shows a possible application of pri: computing revenue (rev) collected from households via a set of primary factor taxes (tax) (differences from earlier figures are shown in bold). For clarity, v has been replaced by $vpre$ to indicate that it is the value before taxes. The notation $vpre(pri)$ in the revenue equation indicates that only the elements of $vpre$ that have subscripts in pri are to be used in this context. In effect, $vpre(pri)$ is a subset of $vpre$ defined over $households$ and pri. As a result, $vpre(pri)$ is compatible in multiplication with tax, which is defined also over pri.

Figure 6: Example Use of a Subset

```
set time (t0,t1,t2,t3,t4,t5) ;
set households (ne,se,nw,sw) ;
set inputs (k,l,e,m) ;
set pri = inputs - (e,m) ;

variable p(inputs) ;
variable q(households, inputs) ;
variable vpre(households, inputs) ;
variable tax(pri) ;
variable rev(households, pri) ;

vpre = p*q ;
rev = tax*vpre(pri) ;
```

Sym also allows this kind of restriction to be imposed by including one or more set names as qualifiers before the equation. For example, the revenue equation could have been given as:

(13) $pri{:}\, rev = tax * vpre$;

The initial pri: indicates that Sym should treat all variables in the equation that are defined over supersets of pri (that is, defined over $inputs$) as though they were restricted to the pri domain via the (pri) notation. This form is more convenient in some circumstances. For example, the following equation would cause the interest rate (int) to be equal to the time preference rate ($timepref$) in the model's last time period (the set $last$ will be discussed below), which would typically be the steady state:

(14) $last{:}\, int = timepref$;

As a variable, the interest rate would be defined over all time periods. However, (14) would only apply in the last period.

Support for set aliases to remove ambiguities:

In some circumstances it is necessary to use a single underlying set for two or more dimensions of a variable. To eliminate the ambiguities that would otherwise arise, Sym provides a mechanism known as "set aliases". Aliases are defined and used much like subsets but Sym treats them as distinct from one another and not interchangeable for the purpose of checking conformability or generating code. Aliases, in effect, are used to indicate a narrower interpretation of the set.

As an example application, suppose a model has six goods (set $sectors$) and five regions (set $regions$). In principle, each region could import each of the goods from each of the regions. If the quantity of imports is imp, it would need to be defined over three sets: one ranging over goods, one ranging over countries of origin, and one ranging over countries of destination. However, the following declaration would be ambiguous because $regions$ is playing two separate roles:

(15) $variable\ imp(sectors, regions, regions);$

To avoid this, aliases for $regions$ can be created and used to resolve the ambiguity. Figure 7 gives an example that uses two aliases, $orig$ and $dest$, to carry out a calculation of tariff revenue by sector ($revtar$) that correctly accounts for export prices at each good's country of origin (pex), and import tariffs at each good's country of destination (tar).

Figure 7: Examples of Set Aliases

```
set time (t0,t1,t2,t3,t4,t5) ;
set sectors (s1,s2,s3,s4,s5,s6) ;
set regions (us,can,mex,eu,row) ;

set orig = regions ;
set dest = regions ;

variable tar(sectors,regions) ;
variable pex(sectors,regions) ;
variable impt(sectors,orig,dest) ;
variable revtar(sectors,regions) ;

revtar = tar*sum(orig,pex(orig)*impt) ;

variable expt(sectors,regions) ;

expt = sum(dest,impt) ;
```

The equation for $revtar$ implies that revenue collected by the US on imports of good $s1$ from all other countries would be computed as:

(16) $revtar_{s1,us} = tar_{s1,us} \sum_o (pex_{s1,o} * imp_{s1,o,us})$

In the absence of ambiguity, variables defined over aliases are conformable with variables defined over the original sets. For example, in the calculation of exports ($expt$) in Figure 7, the result of the sum is defined over $sectors$ and $orig$. However, since $orig$ is an alias for $regions$, and $expt$ is defined over $sectors$ and $regions$, there is no ambiguity and the sum is conformable with $expt$.

Replicating variables across additional domains:

Finally, in some circumstances conformability may require explicitly replicating a variable across an additional domain. For example, some calculations involving the export prices from the previous example may require an export price for each destination country even though export prices only vary by country of origin, not by destination. In that case, conformability can be achieved by using the notation $pex\#dest$, which indicates that all destination regions in $dest$ use the same variable pex. Roughly speaking, it is equivalent to duplicating pex for each destination country.

Summary:

Overall, Sym preserves the clarity of matrix notation but avoids its liabilities by imposing strict conformability rules. By doing so, it is able to determine an unambiguous alignment of dimensioned variables in any operation without regard to the ordering of the variables and without the need for explicit subscripts. The result is that models can be expressed very compactly and many potential opportunities for coding errors are eliminated.

2.4 Planned Modules for Open IGEM

At present, the envisioned structure of Open-IGEM follows the IGEM Fortran code. Thus, there are planned modules for producers, households, investment, governments, and trade. Partitions of these may arise during development as a means of reducing complexity, increasing transparency, or isolating components for possible substitution or model linkage. There will be an intra-temporal module that unifies these within-period strands and an inter-temporal module that governs IGEM's dynamics and foresight.

3 Implementation in GEMPACK

Sym will be used to produce a set of files that implement the model in TABLO, the native modeling language used by GEMPACK. GEMPACK will then be used to solve the model. Sym will be used as the front end language rather than implementing the model directly in TABLO in part because Sym's notation is much more concise. For example, Figure 8 shows the earlier example calculation in both Sym and TABLO.

Figure 8: Example Model in Sym and TABLO

Sym

```
set time (t0,t1,t2,t3,t4,t5) ;
set households (ne,se,nw,sw) ;
set inputs (k,l,e,m) ;

variable p(inputs) ;
variable q(households, inputs) ;
variable v(households, inputs) ;
variable vby(households) ;
variable von(inputs) ;

v = q*p ;
vby = sum(inputs, v) ;
von = sum(households, v) ;
```

TABLO

```
set households (ne,se,nw,sw) ;
set inputs (k,l,e,m) ;
set (intertemporal) time (t0,t1,t2,t3,t4,t5) ;

variable (all,i,inputs) (all,t,time) p(i,t) ;
variable (all,h,households) (all,i,inputs) (all,t,time) q(h,i,t) ;
variable (all,h,households) (all,i,inputs) (all,t,time) v(h,i,t) ;
variable (all,h,households) (all,t,time) vby(h,t) ;
variable (all,i,inputs) (all,t,time) von(i,t) ;

equation EQN1 (all,h,households) (all,i,inputs) (all,t,time)
   v(h,i,t) = q(h,i,t)*p(i,t) ;

equation EQN2 (all,h,households) (all,t,time)
   vby(h,t) = sum(i,inputs,v(h,i,t)) ;

equation EQN3 (all,i,inputs) (all,t,time)
   von(i,t) = sum(h,households,v(h,i,t)) ;
```

However, using Sym as the front-end language has additional benefits as well. It implements the conformability restrictions discussed above to be imposed and is not susceptible to errors regarding the order of subscripts. In addition, it allows the model to be moved to a different solution package very easily, if needed.

GEMPACK is well established in the general equilibrium modeling community. It is widely available, robust, well-supported by the Centre of Policy Studies at Victoria University, has a large user base (including the GTAP project at Purdue University), and has a proven track record over the last 25 years. Moreover, it satisfies the design criteria for Open IGEM: (1) it is scrupulously well documented and its solution procedure is transparent; (2) it is highly flexible and allows endogenous and exogenous variables to be switched easily during individual simulations with no recoding of the model required; (3) it is reliable and requires little or no adjustment of the solution procedure by users, and it is not subject to convergence failures on correctly specified models; and (4) it has high capacity and is suitable for large models.

Although it has many strengths for general equilibrium modeling, GEMPACK's solution algorithm does impose one limitation on Open IGEM. Internally, GEMPACK represents intertemporal models as a large block-diagonal system of equations, with one block for each time period. Although the system is very sparse, and GEMPACK uses sparse matrix algorithms throughout, the memory and time required to solve the model will rise somewhat faster than in proportion to the number of periods. IGEM itself is typically solved at a one-year frequency over a 100 year interval but we expect that a 100-period solution in GEMPACK will take more time to compute than most users will be willing to wait. It will therefore be necessary to use fewer time periods and solve the model at something other than annual frequency.

We do not, however, think this is a significant limitation. Many other models are solved with periods more than one year apart. Moreover, it is possible to produce a highly accurate solution using a small number of periods distributed strategically over the interval to be modeled. A first-order Taylor Series approximation shows that the error introduced in a given variable by spacing periods more than one year apart will depend on the variable's second derivative with respect to time and the number of years between the periods. As a result, an accurate solution can be obtained by spacing periods far apart during years when derivatives of variables in the model will be changing slowly, which is typically the later years of the simulation. For example, a model might be solved over a 100 year interval by including ten periods set in the years 0, 2, 4, 6, 10, 20, 35, 50, 75, 100.

Finally, although GEMPACK is probably best known for solving linearized models quickly, it has a suite of features that allow it to compute nonlinear solutions as well. There is also a well-established boot-strapping procedure for building initial solutions (i.e., baselines) for nonlinear models.

4 Deployment

The completed implementation of Open IGEM will be made available via GitHub, one of the most well-established and widely used platforms for open source projects. Full source code for the Sym program (written in C) will be provided, as well as the Sym files for the model, data files containing the model's parameters and exogenous variables, and utility programs written in Python. Posting the model on GitHub will allow it to be downloaded by anyone, and will also allow users to create their own derivative models by forking the project. In addition, GitHub provides a robust and transparent mechanism for reviewing and incorporating revisions. This mechanism will make the evolution of the model completely transparent to outside users, and will also enable those users to make suggestions that could be incorporated into the model by the IGEM team.