OSCEF: The Open-Source CIM-EARTH Framework User Manual for Version 1.0*

Sou-Cheng T. Choi† Todd Munson‡

March 11, 2014

Argonne National Laboratory Technical Report ANL/MCS-TM-339

*This work was supported by the U.S. National Science Foundation Decision Making Under Uncertainty Program under grant SES-0951576 and by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under contract DE-AC02-06CH11357.

†Computation Institute, University of Chicago, Chicago, IL 60637; e-mail: sctchoi@ci.uchicago.edu.

‡Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439; e-mail: tmunson@mcs.anl.gov.

Contents

List of Figures

List of Tables

Open Source License

1 Introduction

Computable general equilibrium (CGE) models [6, 8, 11] are used when studying the impacts of economic activity on climate change and vice versa. These models typically include many countries, each endowed with different amounts of labor and natural resources; the commodities produced by those countries; and international trade and transportation of the commodities among the countries. These models are extended to account for the greenhouse gases emitted from burning fossil fuels. The Open-Source CIM-EARTH Framework (OSCEF) is an open-source framework for formulating large-scale CGE models [6, 8, 11] based on the CIM-EARTH architecture [4]. OSCEF Version 1.0 includes basic processing of social accounting matrices (SAMs), which contain critical input data for CGE models. Section 2 provides a quick installation guide OSCEF and the third-party libraries, as well as execution of the OSCEF test suite and use cases. Section 3 describes the SAM data format and Section 4 highlights the design and usage of a few key application programming interfaces (APIs) in OSCEF.

2 Quick Start

2.1 Download and Install

Following are the basic steps for installing OSCEF on a UNIX platform running the `bash` shell. For Windows users, we recommend installing Cygwin (`http://www.cygwin.com`) and GNU g++. The steps below are then directly applicable. If you have an account with the Computation Institute (CI) at the University of Chicago, you can install and run OSCEF in your account; refer to Appendix A for details.

2.1.1 Before Installation

1. Check that the following amount of disk space is available in the machine on which you intend to install OSCEF.
   (a) OSCEF version 1.0: 150 MB
   (b) Boost C++ library version 1.54.0: 488.5 MB
2. Check that at least 20 MB of memory are available in the machine for running unit tests.

2.1.2 Installation of OSCEF

1. Download OSCEF source.
   (a) The current release can be obtained from

   `http://www.rdcep.org/oscef`

   (b) The release can also be obtained from the RDCEP subversion repository by running the command

   ```
   svn checkout http://svn.ci.uchicago.edu/svn/rdcep-public/oscef/oscef-1.0
   ```

   the first time; the `svn update` command can then be used to obtain the latest changes in the repository. More details are available at `http://bit.ly/I2rRYp`.

   A subdirectory named `oscef-1.0` containing the OSCEF source files will be created in the current directory. All subsequent directory paths are relative to `oscef-1.0` unless absolute paths are provided or otherwise indicated.
2. Download the Boost C++ library from `boost.org`. Edit your `.bash_profile` to configure the path to the Boost library. The following example assumes that the BOOST C++ library version 1.54 is installed in the home directory.

   ```
   export BOOST_LIB_PATH=~/boost_1_54_0
   ```

Then run the following command at the prompt.

```
source ~/.bash_profile
```

3. (Optional) If you would like to run OSCEF's unit tests (see Step 6), you need to install the CUTE version 1.7.0 standalone library from http://cute-test.com/projects/cute/wiki/CUTE_standalone. Edit your `.bash_profile` to configure the path to the CUTE library. The following example assumes that the CUTE library is installed in the home directory.

```
export CUTE_PATH=~/cute1_7_0
```

Then run the following command at the prompt.

```
source ~/.bash_profile
```

4. (Optional) If you would like to execute OSCEF with a detailed runtime log, you need to install the LOG4CPP library from http://sourceforge.net/projects/log4cpp/files. Edit your `.bash_profile` to configure the path to the LOG4CPP library. The following example assumes that the LOG4CPP library is installed in the home directory.

```
export LOG4CPP_PATH=~/log4cpp
```

Then run the following command at the prompt.

```
source ~/.bash_profile
```

5. (Optional) The OSCEF API documentation in both HTML and PDF is found in the subdirectory `doc`. To regenerate the documentation, you need to install Doxygen from http://www.stack.nl/~dimitri/doxygen.

6. (Optional) To run OSCEF's unit tests in `Test.cpp`, install the CUTE library (see Step 3), go into the subdirectory `tests`, and execute the shell script `Test.sh`. If the installation of OSCEF has been successful, "success" should be the first word in every line of output before the last. the last few lines of the output should look similar to the following.

```
11655:#success CgeTest::operator() OK
12178:#success CgeTest::cgeConstructorTest OK
14048:#success CgeTest::cgeWriteMcpTest OK
14056:~~~~~ End of all OSCEF unit tests ~~~~~
```

2.1.3 Third-Party Data Required

OSCEF uses social accounting matrices (SAMs) from the Global Trade Analysis Project (GTAP) version 7.1 database. Users need to subscribe to the database to obtain the SAMs; see https://www.gtap.agecon.purdue.edu/databases for details. The default location for installing the SAMs is `../datasets/gtap-7.1`. Regarding the structure of and OSCEF operations on SAMs, see Section 3.

2.2 Running a Complete Test Case

The subdirectory `useCases` contains the OSCEF test cases that you can examine, compile, and run. The following three programs, for example, are used to generate data for the "BTA 16x16" model in [4, Section 4.2]. These programs require SAMs from GTAP; see Section 2.1.3 for information.

Program 1. `oscefAggSec` aggregates GTAP 7.1 SAMs from 57 to 16 sectors following the mapping defined in `sector_merger.csv` and outputs the aggregated SAMs in `useCases/outS16`.

Program 2. `oscefAggFac` reads the sector-aggregated files, aggregates the five factors into four according to the mapping defined in `factor_merger.csv` and outputs the aggregated SAMs in `useCases/outS16F4`.

Program 3. `oscefAggReg` reads the factor-aggregated files, aggregates from 112 to 16 regions using the mapping defined in `region_merger.csv` and outputs the resultant SAMs in `useCases/outS16F4R16`.

The following are key steps to compile and execute this use case.

Step 1. Change to the `useCases` directory.

Step 2. To compile and run the sectoral aggregation program:

(a) Compile the file `oscefAggSec.cpp` by running the following at the prompt.

```
make oscefAggSec
```

(b) Run `oscefAggSec` by providing up to three input arguments. When all or some of the arguments are missing, the program provides default values.

i. The first argument is the directory containing the GTAP SAMs input.
ii. The second argument is the OSCEF mapping for sectoral aggregation.
iii. The third argument is the directory for the aggregated SAMs output.

For example, you can run the following command.

```
./oscefAggSec "../../datasets/gtap-7.1/" "../data/bta16x16/sector_merger.csv" "outS16/"
```

The command will produce aggregated files in the subdirectory `useCases/outS16`.

Step 3. To compile and run the factoral aggregation program:

(a) Compile the file `oscefAggFac.cpp` by running the following at the prompt.

```
make oscefAggFac
```

(b) Run `oscefAggFac` by providing up to three input arguments. When all or some of the arguments are missing, the program provides default values.

i. The first argument is the directory containing the SAMs produced by `oscefAggSec`.
ii. The second argument is the OSCEF mapping for factoral aggregation.
iii. The third argument is the directory for the aggregated SAMs output.

For example, you can run the following command.

```
./oscefAggFac "outS16/" "../data/bta16x16/factor_merger.csv" "outS16F4/"
```

The command will produce aggregated files in the subdirectory `useCases/outS16F4`.

Step 4. To compile and run the regional aggregation program:

(a) Compile the file `oscefAggReg.cpp` by running the following at the prompt.

```
make oscefAggReg
```

(b) Run `oscefAggReg` by providing up to three input arguments. When all or some of the arguments are missing, the program provides default values.

i. The first argument is the directory containing the SAMs produced by `oscefAggFac`.
ii. The second argument is the OSCEF mapping for regional aggregation.
iii. The third argument is the directory for the aggregated SAMs output.

For example, you can run the following command.

```
./oscefAggReg "outS16F4/" "../data/bta16x16/region_merger.csv" "outS16F4/"
```

The command will produce aggregated files in the subdirectory `useCases/outS16F4R16`.

Alternatively, `oscefAgg` combines all three aggregation steps in a single program. The C++ source file can be compiled by running the following at the prompt.

```
make oscefAgg
```

Run `oscefAgg` by providing up to three input arguments. When all or some of the arguments are missing, the program provides default values.

1. The first argument is the directory containing the GTAP SAMs input.
2. The second argument is the directory containing the OSCEF mappings for regions, factoral, and sectoral aggregation.
3. The third argument is the directory for the aggregated SAMs output.

For example, you can run the following command.

```
./oscefAgg "../../datasets/gtap-7.1/" "../data/bta16x16/" "outS16F4R16/"
```

The command will produce aggregated files in the subdirectory `useCases/outS16F4R16`.

2.3 Modifying a Test Case

Users can modify and run their own test cases by changing the mapping files for aggregation. For example, the default `oscefAgg` aggregates GTAP 7.1 SAMs from 57 to 16 sectors, 5 to 4 factors, and 112 to 16 regions using the mappings in `../data/bta16x16`. To further aggregate the SAMs into only two regions, you can use the following steps.

1. Create a mapping file with the final 16 regions from the BTA case in the first column, mapping "USA" to "USA", and mapping the other 15 regions to "ROW" (rest of world) in the second column.
2. Run the `oscefAggReg` command using the new mapping file. For example, you can run the following command.

   ```
   ./oscefAggReg "outS16F4R16/" "tworeg.csv" "outS16F4R2/"
   ```

   where `tworeg.csv` is the file containing the new region mapping. The command will produce aggregated files in the subdirectory `useCases/outS16F4R2`.

Another exercise is to repeat the use case in Section 2.2 but aggregate all sectors into 10 (and all marginal commodities into one), five factors into four, and all regions into 10 using the mappers in `../data/bta10x10/`. For example, you could run the following command:

```
./oscefAgg "../../datasets/gtap-7.1/" "../data/bta10x10/" "myoutS10F4R10/"
```

This will store the output from this aggregation in the `myoutS10F4R10`. Since GTAP allows distribution of SAMs with up to 10 sectors and regions, we include the aggregated SAMs in `outS10F4R10` for reference. There should be no difference between the reference SAMs in `outS10F4R10` and the generated ones in `myoutS10F4R10`.

2.3.1 Getting Help

If your installation fails and you need help, please first attempt the following steps and then provide us with the requested information.

1. You are attempting to run the OSCEF unit tests in Step 6 from the `tests` directory.
   (a) Run `make clean` at the prompt.
   (b) Run the following command at the prompt.

   ```
   ./Test.sh > Test.log 2>&1
   ```

   Compare your `Test.log` with the reference `Test-ref.log` contained in the distribution.
   (c) Send rdcep-support@ci.uchicago.edu the files `Test.log`, `Test-stdout.log`, `Test-stderr.log`, and `Test-make-stderr.log` if any of them exists.
2. You are attempting to run the OSCEF use cases from the `useCases` directory (see Section 2.2 for more details).
   (a) Run `make clean` at the prompt.
   (b) Run the following command at the prompt.

   ```
   ./oscefAggSec "../../datasets/gtap-7.1/" > oscefAggSec.log 2>&1
   ```

   Compare your `oscefAggSec.log` with the reference `oscefAggSec-ref.log` contained in the distribution.
   (c) Send rdcep-support@ci.uchicago.edu your file `oscefAggSec.log` if it exists.
3. Other information to include in your email:
   (a) What operating system and version are you using?
   (b) What version of OSCEF are you using?
   (c) Any additional information such as a workaround or suggested solution?

3 Social Accounting Matrices

We represent the economic data input to the CGE models using social accounting matrices and standardize on the particular format output by GTAP version 7.1. Here we discuss the format and contents as well as the transformations applied to produce different regional and sectoral aggregations.

3.1 Data Representation

The economic data consists of a set of files, one file per region, stored within a single directory. Each file contains a table with the economic data for the region in a column separated values format. These files are labeled as SAM_[region code]_[year].csv (e.g. SAM_USA_2004.csv).

The size of each table is a function of s, f, r, m, and c: s is the number of sectors; f is the number of factors; r is the number of regions; m is the number of transportation sectors, typically air, land, sea; and c is the number of capital goods. The values for the various GTAP versions are found in Table 1. Each table is sparse and the nonzeros occur in specific submatrices. In particular the table can be partitioned into 19×19 submatrices, many of which are zero. Figure 1 shows the submatrix structure, in which the nonzero regions are colored. The nonzero submatrics contain information related to expenditures, taxes and subsidies, and international trade. The nonzero submatrices can be grouped into roughly eight categories, as summarized in Table 2. The indices for the submatrices in the overall matrix can be defined in terms of s, f, r, m, and c. We list the beginning indices of each submatrix and their headers in Table 3. The headers are tagged with

Table 1: SAM dimensions defined by s, f, r, m, and c in various versions of GTAP database. OSCEF 1.0 uses GTAP version 7.1.

Description	GTAP Code	Version 5.3/5.4 Year 1997	Version 6.0 Year 2001	Version 7.0 Year 2004	Version 7.1 Year 2004	Version 8 Year 2007
Aggregated Sectors	s	57	57	57	57	57
Aggregated Factors	f	5	5	5	5	5
Aggregated Regions	r	78	85	113	112	129
Margin Commodities	m	3	3	3	3	3
CGDS (capital goods)	c	1	1	1	1	1

{S}, {R}, {F} and {M}, which represent code names for sectors, regions, factors, and margins, respectively, and I and J indicate the index for the block. The information in a SAM is very rich. For example, if we consider the submatrix $T_{4,3}$ "Factor demand," the sum represents the GDP for the region. In the rest of this subsection, we present two formats that OSCEF used for reading, writing, and processing a SAM.

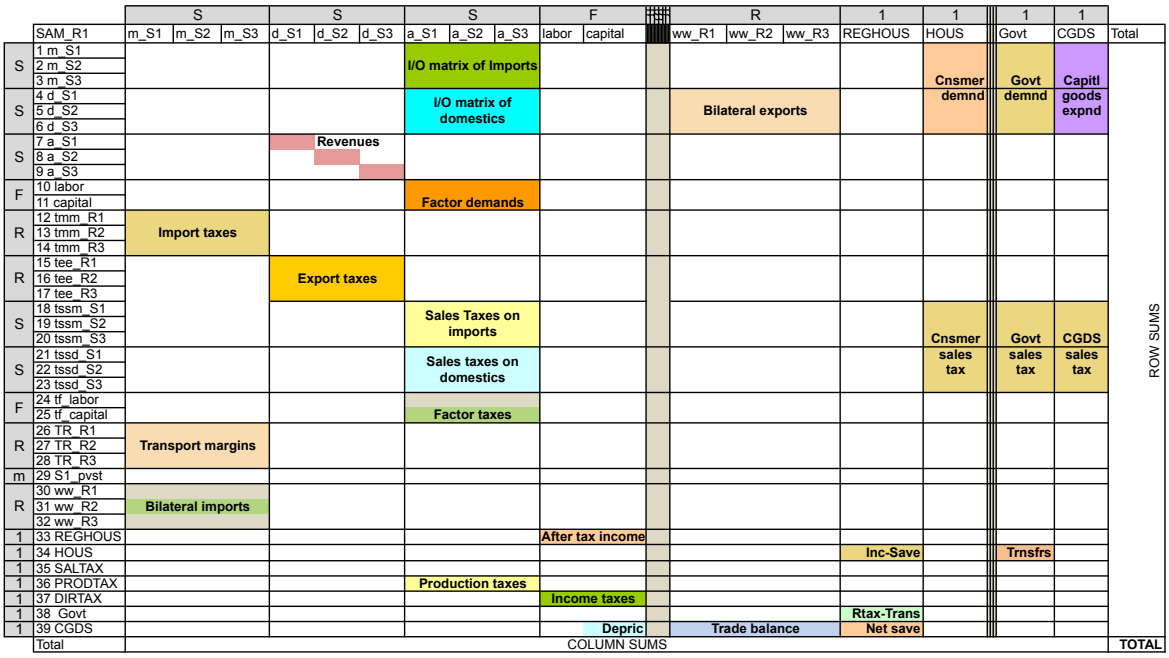

Figure 1: Key for submatrix structure of the SAM.

3.1.1 Dense Matrix Format

For GTAP version 7.1, the overall tables is a 978×978 matrix, not including the row and column headers. The metadata of the dense format is described in Table 4. OSCEF requires SAMs to be stored as text in csv (comma-separated values) files. The first row contains the text label "SAM" and column headers, finishing with the text label "Total" in the last entry. Then we have row headers followed by the matrix values itself, with the last column being row sums. The last row starts with text label "Total" again, followed by column sums and a final entry of total matrix sum.

Table 6 is an instance of an aggregated SAM with $s = 4$, $f = 2$, $r = 3$, and $m = 1$. To save space, we have omitted rows 4 to 43, some zero entries in rows 1-3 and 44, and some entries in the last row of column sums; the omissions are indicated by ellipses (...).

Table 2: Keys of nonzero blocks in a SAM.

$T_{I,J}$	**Key**	**Values**
Domestic Commodity Producers (including government services)		
$T_{1,3}$	Expenditure on commodities from Armington importers	+
$T_{2,3}$	Expenditure on commodities from domestic producers	+
$T_{4,3}$	Expenditure on factors from domestic consumers	+
$T_{7,3}$	Taxes paid on commodities from Armington importers	+/−
$T_{8,3}$	Taxes paid on commodities from domestic producers	+/−
$T_{9,3}$	Taxes paid on factors from domestic consumers	+/−
$T_{16,3}$	Taxes paid on revenue	+/−
Domestic Investment Production		
$T_{1,19}$	Investment expenditure on imported commodities	+
$T_{2,19}$	Investment expenditure on domestic commodities	+
$T_{7,19}$	Investment taxes paid on imported commodities	+/−
$T_{8,19}$	Investment taxes paid on domestic commodities	+/−
$T_{16,19}$	Investment taxes paid on revenue	+/−
Armington Importers		
$T_{12,1}$	Armington importer bilateral import expenditure	+
$T_{10,1}$	Armington importer homogeneous transport expenditure	+
$T_{5,1}$	Armington taxes on imports paid to importing country	+/−
$T_{6,2}$	Armington taxes on imports paid to exporting country	+/−
Homogeneous Transport		
$T_{2,11}$	Domestic transportation exported to homogeneous transport	+
Government Consumer		
$T_{1,18}$	Government expenditure on imported commodities	+
$T_{2,18}$	Government expenditure on domestic commodities	+
$T_{7,18}$	Government tax on imported commodities	+/−
$T_{8,18}$	Government tax on domestic commodities	+/−
$T_{18,13}$	Government income from taxes	+
Private Consumer		
$T_{1,14}$	Consumer expenditure on imported commodities	+
$T_{2,14}$	Consumer expenditure on domestic commodities	+
$T_{7,14}$	Consumer tax on imported commodities	+/−
$T_{8,14}$	Consumer tax on domestic commodities	+/−
$T_{19,4}$	Capital depreciation	+
$T_{19,11}$	Homogeneous transport trade imbalance	+/−
$T_{19,12}$	Trade imbalance	+/−
$T_{19,13}$	Net consumer expenditure on investment	+
$T_{13,4}$	After tax revenue from consumer factors	+
$T_{17,4}$	Consumer taxes paid on consumer factors	+/−
$T_{19,4}$	Consumer depreciation paid on consumer factors	+
$T_{14,13}$	Consumer expenditure on products	+
Tax Accounts		
$T_{13,5}$	Taxes collected on import duties	+/−
$T_{13,6}$	Taxes collected on export duties	+/−
$T_{13,7}$	Taxes collected on imported commodities	+/−
$T_{13,8}$	Taxes collected on domestic commodities	+/−
$T_{13,9}$	Taxes collected on consumer factors paid by producers	+/−
$T_{13,16}$	Taxes collected from producer revenue (including investment)	+/−
$T_{13,17}$	Taxes collected on consumer factors paid by consumers	+/−
Other Accounts		
$T_{3,2}$	Total producer revenue	+
$T_{11,10}$	Total homogeneous transport expenditures for the region	+
$T_{2,12}$	Total domestic commodities expenditures exported	+

Table 3: First indices and header names of submatrices in GTAP SAM version 7.1.

I, J	**First Indices**	**Header**	I, J	**First Indices**	**Header**
1	1	m_{S}	11	$5s+2f+2r+mr+1$	{M}_pvst
2	$s+1$	d_{S}	12	$5s+2f+2r+mr+m+1$	ww_{R}
3	$2s+1$	a_{S}	13	$5s+2f+3r+mr+m+1$	REGHOUS
4	$3s+1$	{F}	14	$5s+2f+3r+mr+m+2$	HOUS
5	$3s+f+1$	tmm_{R}	15	$5s+2f+3r+mr+m+3$	SALTAX
6	$3s+f+r+1$	tee_{R}	16	$5s+2f+3r+mr+m+4$	PRODTAX
7	$3s+f+2r+1$	tssm_{S}	17	$5s+2f+3r+mr+m+5$	DIRTAX
8	$4s+f+2r+1$	tssd_{S}	18	$5s+2f+3r+mr+m+6$	Govt
9	$5s+f+2r+1$	tf_{F}	19	$5s+2f+3r+mr+m+7$	CGDS
10	$5s+2f+2r+1$	{M}_{R}			

3.1.2 Sparse Matrix Format

Since most SAM matrices contain a lot of zeros ($\geq$ 70% of all entries), we could use sparse matrices for storage to reduce memory usage by 50% or more. The first line in a sparse-SAM csv file is the same as that in a dense-SAM file. It is then followed by each *nonzero* row of a SAM, starting with the row index, and pairs of column indices and nonzero entries in the row. The last two lines in the csv file are dense and contain row sums, column sums, and matrix sum as depicted in Table 5. Table 7 is the same example SAM from Section 3.1.1 except that it is defined in sparse format.

3.2 Transformation Tools

In this section, we summarize a few standard operations commonly performed on a SAM or a set of SAMs. These operations include symmetric permutations; renaming header codes; and aggregation of SAMs by sectors, factors, regions, and marginal commodities. In each operation, a sequence of orthogonal updates on social accounting matrix S is performed such that $S \leftarrow Q^T S Q$, where Q is a square or rectangular orthogonal matrix; that is, $Q^T Q = I$, where I is the identity matrix. A list or a mapper is specified by the OSCEF user for constructing Q. The list or mapper can be defined by an external text `csv` file with one or two columns or can be programmatically created by the C++ classes `List` or `Mapper` in OSCEF.

The basic steps for constructing Q are to select relevant block indices (I, J), where $1 \leq I, J \leq 19$, and $T_{I,J} \neq 0$ so that $Q^T S Q$ actually is applied only on submatrix $T_{I,J} \leftarrow U^T T_{I,J} V$ for some orthogonal matrices U and V; that is, $Q^T S Q$ has no effect on other subblocks $T_{K,L}$ for $K \neq I$ and $L \neq J$. The choices of (I, J) depends on whether the operation is related to sectors, factors, regions, or marginal commodities. In the following subsections, we describe the specific details of a list or a mapper, the orthogonal matrices, and their defining indices for each class of operations.

Table 4: Metadata of a SAM in dense format.

SAM	[Column headers]	Total
[Row headers]	[Dense matrix]	[Row sums]
Total	[Column sums]	[Matrix sum]

Table 5: Metadata of a SAM in sparse format.

SAM SPARSE	[Column headers]	Total
	[Sparse matrix]	
Total	[Row sums]	
Total	[Column sums]	[Matrix sum]

Table 6: SAM in dense format.

```
SAM,1 m_ALL,2 m_DWE,3 m_TRA,4 m_GOV,5 d_ALL,6 d_DWE,7 d_TRA,8 d_GOV,9 a_ALL,10
    a_DWE,11 a_TRA,12 a_GOV,13 labor,14 capital,15 tmm_USA,16 tmm_ROW,17 tmm_DEV
    ,18 tee_USA,19 tee_ROW,20 tee_DEV,21 tssm_ALL,22 tssm_DWE,23 tssm_TRA,24
    tssm_GOV,25 tssd_ALL,26 tssd_DWE,27 tssd_TRA,28 tssd_GOV,29 tf_labor,30
    tf_capital,31 TRA_USA,32 TRA_ROW,33 TRA_DEV,34 TRA_pvst,35 ww_USA,36 ww_ROW
    ,37 ww_DEV,38 REGHOUS,39 HOUS,40 SALTAX,41 PRODTAX,42 DIRTAX,43 Govt,44 CGDS,
    Total
1 m_ALL,0,...,0,791361,2413,22388,56489,0,...,0,464855,0,0,0,1388,240315,1579208
2 m_DWE,0,...,0
3 m_TRA,0,...,0,30156,0,13896,4074,0,...,0,16689,0,0,0,375,2,65191
...
44 CGDS,0,...,0,1045522,0,...,0,31859,0,316548,219544,584979,0,...,0,2198452
Total,1579208,0,65191,37171,14911409,1191313,...,2198452,80902362
```

Table 7: SAM in sparse format.

```
SAM SPARSE,1 m_ALL,2 m_DWE,3 m_TRA,4 m_GOV,5 d_ALL,6 d_DWE,7 d_TRA,8 d_GOV,9
    a_ALL,10 a_DWE,11 a_TRA,12 a_GOV,13 labor,14 capital,15 tmm_USA,16 tmm_ROW,17
     tmm_DEV,18 tee_USA,19 tee_ROW,20 tee_DEV,21 tssm_ALL,22 tssm_DWE,23 tssm_TRA
    ,24 tssm_GOV,25 tssd_ALL,26 tssd_DWE,27 tssd_TRA,28 tssd_GOV,29 tf_labor,30
    tf_capital,31 TRA_USA,32 TRA_ROW,33 TRA_DEV,34 TRA_pvst,35 ww_USA,36 ww_ROW
    ,37 ww_DEV,38 REGHOUS,39 HOUS,40 SALTAX,41 PRODTAX,42 DIRTAX,43 Govt,44 CGDS,
    Total
1,9,791361,10,2413,11,22388,12,56489,39,464855,43,1388,44,240315
3,9,30156,11,13896,12,4074,39,16689,43,375,44,2
...
44,14,1045522,34,31859,36,316548,37,219544,38,584979
Total,1579208,0,65191,37171,14911409,1191313,...,2198452
Total,1579208,0,65191,37171,14911409,1191313,...,2198452,80902362
```

3.2.1 Permutations

Symmetric permutations on a SAM can be on sectors, factors, regions, and margins. A *sector* permutation permutes rows or columns in a block related to sectors, whose row header or column header is tagged with "{S}" in Table 3. The sectoral block index set is thus defined as $\mathcal{I} \equiv \{1, 2, 3, 7, 8\}$, and its complement is $\mathcal{I}^c \equiv \{1, \ldots, 19\} \backslash \mathcal{I}$.

Suppose there are four sectors in an economy, whose codes are `ALL`, `DWE`, `TRA`, and `GOV`. Let them be the row and column headers of $T_{1,3}$ in order. If we want to rearrange the rows and columns of $T_{1,3}$ with the new order `DWE`, `ALL`, `GOV`, and `TRA` instead, OSCEF needs the following input `csv` list file with a column of the sectoral codes in the new order:

```
DWE
ALL
GOV
TRA
```

Alternatively, the list can be programmatically constructed by the C++ class `Sectors`, which is a subclass of `List`, provided by OSCEF. With this list, OSCEF's method `sectorPermute` in the class `Sam` defines a permutation matrix $P = \begin{bmatrix} 0&1&0&0 \\ 1&0&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix}$. Let $T_{I,J} \in \mathbb{R}^{p \times q}$. We have the following four defining cases for U and V

in terms of P, the identity matrices I_p and I_q, and index sets $\mathcal{I}$ and $\mathcal{I}^c$.

$$\begin{cases} U = P, V = I_q & \text{if } I \in \mathcal{I}, J \in \mathcal{I}^c \\ U = I_p, V = P & \text{if } I \in \mathcal{I}^c, J \in \mathcal{I} \\ U = V = P & \text{if } I \in \mathcal{I}, J \in \mathcal{I} \\ U = I_p, V = I_q & \text{if } I \in \mathcal{I}^c, J \in \mathcal{I}^c. \end{cases} \tag{1}$$

When U or V is an identity matrix or $T_{I,J} = 0$, of course we do not carry out the matrix multiplication explicitly. The implementation also permutes the header labels accordingly.

Similar permutations for regions, factors, and marginal commodities can be defined. The differences are in the definition of index set $\mathcal{I}$.

$$\mathcal{I} = \begin{cases} \{1, 2, 3, 7, 8\} & \text{for sectoral operations} \\ \{5, 6, 10, 12\} & \text{for regional operations} \\ \{4, 9\} & \text{for factoral operations} \\ \{1, 2, 3, 7, 8, 10, 11\} & \text{for marginal operations.} \end{cases} \tag{2}$$

Extra care is exercised for operations on a submatrix with row or column index 10, found in regional or marginal permutations. The reason is that it is composed of m blocks of all regions, where m is the number of marginal commodities. Thus, a regional permutation for $T_{10,1}$ is $\text{Diag}(U^T, \ldots, U^T) \times T_{10,1}$, where Diag gives a block-diagonal matrix.

We note that marginal commodities are subsets of sectors. Thus marginal permutations on sectoral submatrices operate only on the relevant rows and columns related to marginal commodities but not to other sectors. While `Sam::sectorPermute` is applicable to permuting marginal commodities, we also have a dedicated method, `Sam::marginPermute`, for the purpose.

3.2.2 Renaming Header Codes

Renaming header codes on a SAM can be selectively performed on sectoral, factoral, regional, or marginal commodity submatrices. Suppose we have only two regions USA and ROW (rest of the world) and that they are associated with the SAM files `SAM_USA_2004.csv` and `SAM_ROW_2004.csv`, respectively. If we want to rename ROW to NUS (not US), then we may define the following mapper in a `csv` file with two columns.

```
USA,USA
ROW,NUS
```

The first column consists of the original regional codes and the second column the new codes. Then OSCEF's interface `sectorPermute` in the C++ class `Sam` defines $U = I_p$ and $V = I_q$ for each $0 \neq T_{I,J} \in \mathbb{R}^{p \times q}$ as a special case of regional permutations defined in the previous subsection (without explicitly carrying out any matrix multiplication with identity), and the implementation changes the header labels accordingly. Since the SAM filenames also contain the regional codes, they will be changed as necessary.

Renaming header labels for sectors, factors, and marginal commodities are similar and treated as special cases of symmetric permutations; see Section 3.2.1. In these cases, however, SAM filenames are not changed as in a regional operation.

3.2.3 Aggregation

Aggregation of SAMs by sectors, factors, regions, or marginal commodities means identifying the subblock index set $\mathcal{I}$ as in (2) and summing the corresponding rows and columns. The operation returns a new SAM that is typically a square matrix of smaller size than the original SAM, but with the same total sum.

Suppose we have three sectors in an economy and their codes are `ALL`, `DWE`, and `TRA`. If we want to combine the first two sectors into one and call the aggregated sector `APT`, OSCEF needs the following input mapper in a `csv` file.

```
ALL,APT
DWE,APT
TRA,TRA
```

The first column is the original sectoral codes, and the second column is their code names after aggregation. Then OSCEF's method `sectorAggregate` in the class `Sam` defines an matrix $P = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. For each nonzero $T_{I,J} \in \mathbb{R}^{p \times q}$, we have U and V defined as in (1) and act on $T_{I,J}$ on its left and right, respectively. For instance, suppose we are given the sectoral submatrix in a SAM $T_{1,3} = \left[\begin{smallmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{smallmatrix}\right]$. To aggregate the first two rows and columns, we update $T_{1,3} \leftarrow U^T T_{1,3} V = P^T T_{1,3} P = \left[\begin{smallmatrix} 12 & 9 \\ 15 & 9 \end{smallmatrix}\right]$.

Aggregating regions, factors, and marginal commodities is similar. For regional aggregation, an additional step sums across all SAM matrices according to the same mapper. The regional codes in the resultant SAMs are also changed as necessary.

Extra care is exercised for operations on a submatrix with row or column index 10, found in regional or marginal aggregation. The reason is that it is composed of m blocks of all regions, where m is the number of marginal commodities. Thus, a regional aggregation for $T_{10,1}$ is $\mathrm{Diag}(U^T, \ldots, U^T) \times T_{10,1}$, where Diag gives a block-diagonal matrix.

We note that marginal commodities are subsets of sectors. Thus, marginal aggregation on sectoral submatrices operate only on the relevant rows and columns related to marginal commodities but not other sectors. While `Sam::sectorAggregate` is applicable to permuting marginal commodities, we also have a dedicated method, `Sam::marginAggregate`, for the purpose.

4 C++ Classes and APIs

An important part of this project is to provide tools for reading, writing, and transforming the static data SAMs for the base year along with corresponding APIs. Figure 2 summarizes the C++ classes we use to model SAMs. We cannot enumerate all the important practices and guiding principles in developing a large scientific framework such as OSCEF. Instead, we refer interested readers to some of the more recent trade books such as [5, 9, 10].

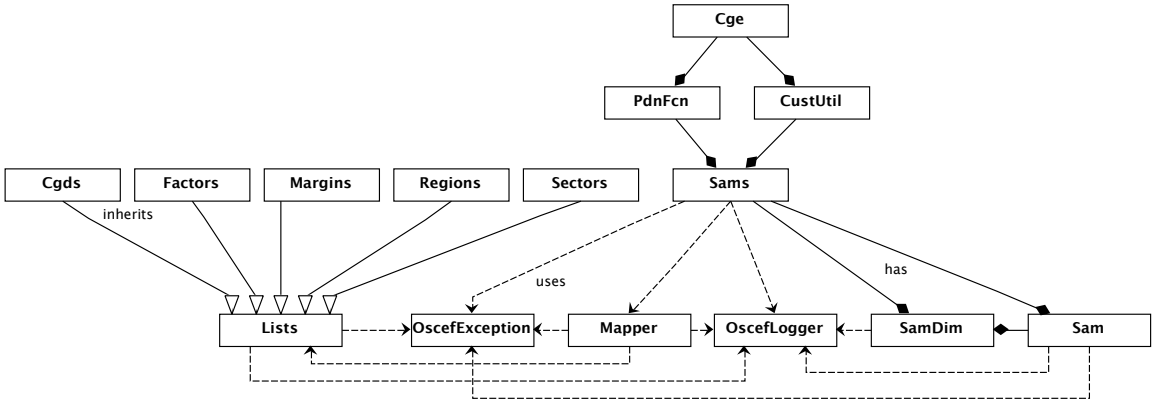

Figure 2: C++ classes in OSCEF.

The following is a list of key APIs in OSCEF version 1.

1. Read and write description of regions and sectors. The following APIs reads in two csv files stored in a directory called "data."

```
Regions regions("../data/regionlist.csv");
Sectors sectors("../data/sectorlist.csv");
```

The format of these csv files are simple; see the following table for an example of a list with six regions.

```
No.,String
1,AUS
2,NZL
3,XOC
4,CHN
5,HKG
6,JPN
```

2. Read and write SAMs. To read a SAM matrix in dense format and write it in sparse format, we can issue the following commands.

```
Sam sam("../data/SAMS_USA_2004.csv");
bool isSparse = true;
sam.write("../output/SAMS_USA_2004.csv", isSparse);
```

For reading and writing multiple SAMs in sparse format, the APIs are similar except that we use the class `Sams` instead of `Sam`. The second and third arguments to the constructor make the program use sparse matrices for storage and read the csv files in sparse format.

```
Sams sams("../data", isSparse, isSparse);
sams.write("../output", isSparse);
```

To have the SAM data-storage formats automatically managed by the program, simply call the following static API before invoking any `Sam` or `Sams` constructors.

```
bool USE_SPARSE = true;
Sam::setAutoSparsify(USE_SPARSE);
```

3. Querying social accounting matrices. To retrieve the value of an entry in a SAM, we can use the following API.

```
int i = 6, j = 4;
double val = sam.getValue(i, j);
```

(a) Sparse storage should be used for producers, along with mechanisms to query the nonzero inputs. The following instance extracts the submatrix $T_{3,2}$ and stores it in a sparse matrix:

```
int I = 3, J = 2;
Double_Sparse_Matrix T = sam.getSparseSubmatrix(I, J);
```

We need to be able to work with these index sets to check consistency later, mainly by way of intersections. Let I be a given index set of columns. Then $S_{\bullet I} = \{S_{\bullet i} | i \in I\}$ can be retrieved as a sparse matrix by using the following API.

```
int I[3] = {1, 2, 5};
Double_Sparse_Matrix T = sam.getSparseColumns(I);
```

(b) Dense storage should be used for bilateral trade among the regions. There are a few components: one is the value of the goods traded, and the other is the transport margins. The following example extracts the submatrix $T_{2,12}$ and stores it in a dense matrix.

```
int I = 2, J = 12;
Double_Matrix T = sam.getSubmatrix(I, J);
```

Likewise, $S_{\bullet I}$ can be retrived as a dense matrix with the following API.

```
int I[3] = {1, 2, 5};
Double_Matrix T = sam.getColumns(I);
```

4. Consistency checks to ensure balances by sector, region, and international trade flows. The following API returns true if each row sum r_i is *numerically* equal [7] to its corresponding column sum c_i, that is, $|r_i - c_i| \leq \epsilon \max(|r_i|, |c_i|)$ for each row $i = 1, \ldots, n$, where n is the size of the SAM and ϵ denotes machine precision.

```
bool bal = sam.isBalanced();
```

If a tolerance parameter is specified, then the API returns true if $|r_i - c_i| \leq \texttt{tol} \times \max(|r_i|, |c_i|)$ for each row $i = 1, \ldots, n$. For instance, if the following example returns true with $\texttt{tol} = 10^{-6}$, then every pair of sums agree to the most significant six digits.

```
double tol = 1e-6;
bool bal = sam.isBalanced(tol);
```

To check if every SAM in a `Sams` instance is balanced, we provide the following API where `tol` is optional and defaults to ϵ.

```
bool bal = sams.areBalanced(tol);
```

5. Aggregation tools for transforming the matrices. To aggregate by regions, we use the following.

```
sams.regionAggregate("../data/region\_merger.txt");
```

The following is an example mapper, also a csv file, in which the first column contains codes of three existing regions in a SAM or multiple SAMs, to be aggregated into a new region called "URD" as listed in the second column of the file.

```
USA,URD
ROW,URD
DEV,URD
```

To aggregate by sectors, we use the following.

```
sam.sectorAggregate("../data/sector\_merger.csv");
```

To aggregate by factors, we use the following.

```
sam.factorAggregate("../data/factor\_merger.csv");
```

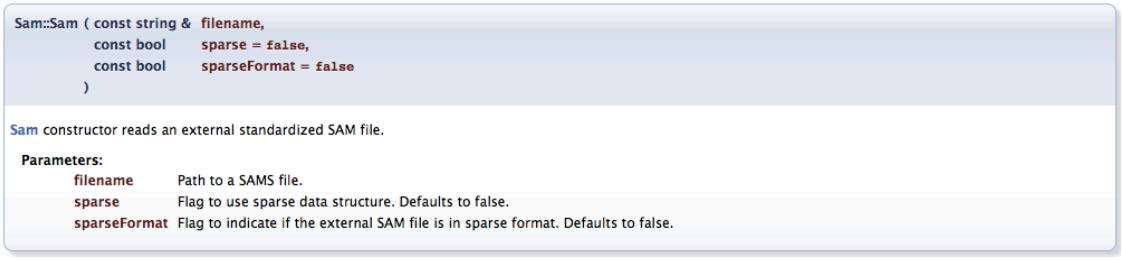

Figure 3: HTML documentation of a SAM constructor generated by Doxygen.

6. Diagnostics. We provide methods `write`, `print`, `isValid` for classes `Sam` and `Sams` so that the data of an instance can be written to an external file or printed to standard output. For example, we have the following.

```
sam.print();
```

7. Logging. Users may take advantage of the third-party package Log4Cpp [3] for logging messages to an external file at various detail levels such as INFO, WARN, DEBUG, and ERROR. The employment of the package is optional at compilation time of OSCEF.

```
OSCEF_DEBUG("Size=%d", sam.getSize());
```

4.1 OSCEF Documentation

In addition to this technical report, we also use the third-party package *Doxygen* [2] for creating online HTML documentation. For example, Figure 3 provides an impression of the Doxygen documentation for the constructor of SAM generated from the following annotated comment. Figure 4 is a partial display of the documentation for the class `List` with an inheritance graph.

```
/**
 * @brief Sam constructor reads an external standardized SAM file.
 *
 * @param filename Path to a SAMS file.
 *
 * @param sparse Flag to use sparse data structure. Defaults to false.
 *
 * @param sparseFormat Flag to indicate if the external SAM file is in sparse
 * format.  Defaults to false.
 */
Sam::Sam(const string& filename, const bool sparse, const bool sparseFormat);
```

4.2 Unit Test Suite

Development and quality of OSCEF APIs is driven by rigorous unit tests built on CUTE [1]. A suite of more than 200 short and longer unit tests are available with the OSCEF source code for verification and extension. The short tests take about ten seconds to execute and the longer tests are optional.

A Installing OSCEF on a CI Machine

If you have an account with the Computation Institute (CI) at the University of Chicago, installation of OSCEF can be simplified.

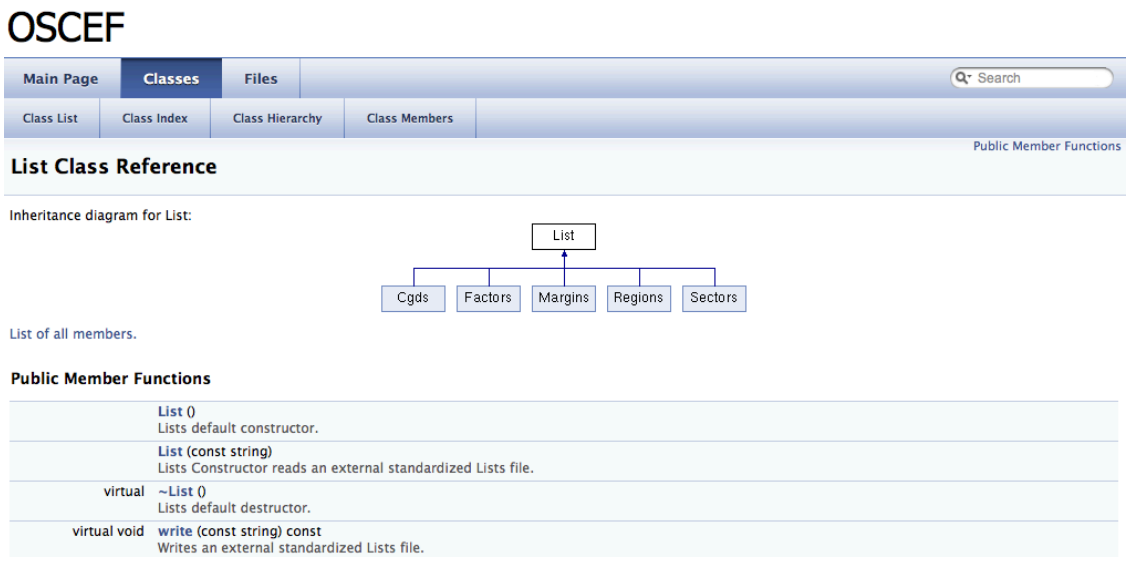

Figure 4: HTML documentation of the class `List` generated by Doxygen.

1. Remote login from your computer to a CI machine using

   ```
   ssh yourUserLogin@login.ci.uchicago.edu
   ```

2. The Boost C++ libraries are already installed on the CI machines and you need to load the appropriate module following the two steps below.

   (a) Add the following to your `.bashrc` file.

   ```
   [ -f /soft/Modules/etc/modules.sh ] && . /soft/Modules/etc/modules.sh
   ```

   Then edit your `.bash_profile` to configure the path to the Boost library.

   ```
   export BOOST_LIB_PATH=$MODULEPATH
   ```

   (b) Run the following commands at the prompt.

   ```
   source ~/.bashrc
   source ~/.bash_profile
   module load boost
   ```

   You will receive the following message if the Boost library is successfully loaded.

   ```
   boost version 1.48.0 (gnu-4.1 compiler) loaded
   ```

3. Follow Steps 3 - 6 in Section 2.1.2.

References

[1] *CUTE: C++ Unit Testing Easier*. http://cute-test.com.

[2] *Doxygen documentation*. http://www.doxygen.org.

[3] *Log for C++ Project*. http://log4cpp.sourceforge.net.

[4] J. Elliott, I. Foster, K. Judd, E. Moyer, and T. Munson, *CIM-EARTH: Framework and Case Study*, The B.E. Journal of Economic Analysis & Policy, 10 (2010).

[5] M. Gregoire, N. A. Solter, and S. J. Kleper, *Professional C++*, John Wiley & Sons, 2011.

[6] L. Johansen, *A Multisectoral Study of Economic Growth*, North Holland, Amsterdam, 1960.

[7] D. E. Knuth, *The Art of Computer Programming. Vol. 2*, Addison-Wesley Publishing Co., Reading, Mass., second ed., 1981. Seminumerical algorithms, Addison-Wesley Series in Computer Science and Information Processing.

[8] S. Robinson, *Macroeconomics, Financial Variables, and Computable General Equilibrium Models*, World Development, 19 (1991), pp. 1509–1525.

[9] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 2003.

[10] ——, *Programming: Principles and Practice Using C++*, no. v. 10 in Developer's Library, Addison-Wesley, 2009.

[11] I. Sue Wing, *Computable General Equilibrium Models and Their Use in Economy-Wide Policy Analysis*, Technical Note 6, Joint Program on the Science and Policy of Global Change, 2004.