**A. Python code to read model-specific submissions (AgMIP3.py)**

1. Open the ‘options’ file. There is one file for each model (see example on next page). The options file contains the following information:
   1. Model name (‘model:’)
   2. Location of submission (‘folder:’)
   3. File name of submission (‘filename:’)
   4. Number of lines to skip at the beginning of the file (‘skip:’)
   5. Column concordance (‘Concordance:’) This option requires seven lines and is described further below.
   6. The column number containing the year (‘Year column:’)
   7. The column number containing the value (‘Value column:’)
   8. The column number containing the units (‘Unit column:’)
   9. (Optional) The field delimiter (‘delimiter:’). The default is a comma.
2. Create the output filename, equal to the input filename plus the current date.
3. Open the input file, skip ‘*n*’ lines depending on the value ‘skip’ in the options file. If skip is ‘-1’, the program assumes that there is not a header row. If skip ‘>=0’, the header row is read and parsed though at the moment the fields are ignored.
4. Each row (i.e., record) is read one at a time. This is to allow for some diagnostic checking of input. These include:
   1. parsing and checking the year
   2. checking for special values such as ‘#DIV/0!’ and ‘N/A’ (in various versions)
   3. substituting some labels for defaults (for example replacing ‘World’ with ‘WLD’)
   4. All labels are quoted
   5. Columns may be permuted depending on the values in the ‘Concordance’
   6. Some scenarios are dropped depending on the values in the ‘Opt’ file
   7. If the row passes all the checks and is not dropped, it is added to the list
5. Input file is closed
6. The list is converted to a Pandas data frame with fixed column names
7. Duplicate records are identified. If there are duplicate rows they will be saved to an Excel and CSV file and are dropped from the data frame.
8. The final set of records is saved as a CSV file with no header.

**Example of an ‘Opt’ file for initial processing of the model-specific submissions**

Model:

AIM

Folder:

V:/AgMIP/AIM

Filename:

AIM\_DIET\_ALL\_18FEB2021

Skip:

0

Concordance (requires 7 lines):

2

3

5

4

6

7

9

Year column:

5

Value column:

7

Unit column:

6

**B. GAMS code to read all submissions and merge into a single data cube (ReadCSV.gms)**

1. Read valid labels for each field: time (t/tt), scenario (scen), model (mod), regions (r), indicator (v), sector or item (i), and units (u). These are contained in a file called ‘AgMIPSets.gms’
2. Read each submission—one at a time. The submissions will be stored in a variable called ‘AgMIPData’ with the following dimensions: AgMIPData(Mod, Scen, r, v, i, t, u). GAMS is strict about label verification. If a read-in label is not part of the labels declared in ‘AgMIPSets.gms’, the program will issue an error and not create the merged cube.
3. If successful, create the merged data cube and store in a GDX container.

**C. GAMS code to extract select records from merged data cube (ReadCSV.gms)**

1. Read valid labels (same as above, i.e., from ‘AgMIPSets.gms’).
2. Read the merged cube from the GDX container.
3. Create regional and indicator filters, optionally all for each or both.
4. Open a CSV file and create the header.
5. Loop over all records in the merged data cube and output only those records consistent with the filter options for regions and indicators.

**D. GAMS code to get unique field labels from merged cube (getSets.gms)**

1. Read valid labels (same as above, i.e., from ‘AgMIPSets.gms’).
2. Read the merged cube from the GDX container.
3. For each field, get the unique set of labels (e.g., regions) and save these to a CSV file (one for each field).

**Potential improvements**

1. The permutation of the columns in the Python code could be simplified and improved. Rather than use numbers, some system could be set up to use labels—either read them in directly from the CSV submission, or include them in the ‘opt’ file. The permutation indices lead to head-aches. We could also drop some of the other options in the file such as the column number for the year, value and unit.
2. Reading by row can also be dropped and some of the validation checks could be simplified.
3. It would nice to be able to check the field labels with a standard set rather than wait till we get to the GAMS code. The Python code could create a list of (unique) field labels and perhaps flag those that do not line up with the current set of ‘standard’ labels. A separate concordance table may allow for swapping bad labels, though ones that are recognized, for example ‘WLD’ for ‘World’, rather than have it hard-coded. This would be especially helpful for scenarios and units where individual teams have been creative. In all cases of using non-standard labels, a warning should be issued and shared with the submitting team.
4. Another wonderful feature would be to have built-in diagnostics for plausibility checking such as valid units, outliers, unusual growth rates, etc. This could be complemented by some visual tools such as box and whisker plots.