PyIO: Input-Output Analysis with Python

By

Suahasil Nazara, Dong Guo,
Geoffrey J.D. Hewings and Chokri Dridi

REAL 03-T-23 October 2003

PyIO

(read: pa**I**-o)

Input-Output Analysis with Python

Suahasil Nazara, Dong Guo, Geoffrey J.D. Hewings, Chokri Dridi

Regional Economics Applications Laboratory, University of Illinois at Urbana-Champaign

Contents

PyIO: Input-output Analysis with Python

Suahasil Nazara, Dong Guo, Geoffrey J.D. Hewings, Chokri Dridi
Regional Economics Applications Laboratory, University of Illinois at Urbana-Champaign

1. Introduction

This note outlines the use PyIO, a Python module for input-output analysis. It is hoped that this module will be the first in a series of modules that address different and more complex methods of analysis based on input-output, social accounting and computable general equilibrium models. What are included in the current module are functions to perform basic operations and analysis with input-output models. The module does not provide the capability for assembling or generating an input-output table; the program assumes that the user has an input-output table available for manipulation. Further, the program assumes familiarity with the basics of input-output analysis; readers needing to refresh their recollection of input-output models are referred to Miller and Blair (1985). In the near future, this manual will be linked to a more extensive text on input-output analysis that is nearing completion (Sonis and Hewings, 2004).

The applications in this module are derived from input-output analysis conducted by colleagues at the Regional Economics Applications Laboratory (REAL), University of Illinois at Urbana-Champaign. The calculations in an input-output framework typically comprise a set of routines. This point warrants the use of functions in the input-output analysis. For a specific analysis, e.g., an extraction method, the computation steps will be identical regardless of input-output of different years or of different dimensions.

Python is chosen as the basis for this module for several reasons. First, Python is easy to learn and has great computational capability. It is also possible to build a graphic user's interface which frees the user from any requirement to know anything about Python itself. At some point, the manual will accompany a program that will be prepared in point-and-click form. Secondly, the core of Python is free. One can download, as well as distribute, all files needed to run this module freely from the internet. This is of great advantage to many, especially to users in developing countries where legal access to software such as Matlab or Stata is limited for various reasons.

This note will outline several functions related to input-output analysis, which are categorized in three parts below:

1. Table operations
   - Aggregation of input-output tables
     - Aggregating sectors in one region (national) or inter-regional input-output tables
     - Aggregating regions in an inter-regional input-output table
   - Updating matrix: regional supply percentage, simple location quotient
   - Updating matrix: RAS method

2. Basic I-O analysis
   - Leontief and Ghoshian inverse
   - Impact analysis
   - Multiplier analysis: output, income, employment, and input

3. Advanced I-O analysis
   - Key sector analysis
   - Output decomposition analysis
   - Multiplier Product Matrix (MPM) analysis
   - Extraction method analysis
   - Push and pull analysis
   - Field of Influence

Besides for the input-output analysis, inputting input-output data is a very important part in this python module. For the sake of programming efficiency, PyIO developed at this stage allows only input from an ASCII text file. This decision was taken to facilitate accessibility to all users of computer. The general structure is that users need to input the information about input-output as a text file. When the computation results are ready to be presented, then users will be asked whether they want those written in Microsoft Excel. If so, Excel needs to be properly installed, and the results will be shown in a worksheet that is automatically opened. If users do not want the output in Excel format, then the results will be automatically written in a pre-named text file.

Aside from the point-and-click environment, it is hoped at a later stage the module would be able to read the data from wider variety of spreadsheet software. As an object oriented programming language, Python has that capability. At present, the current module is also capable only to receive a fairly aggregate input-output structure, i.e., one vector of final demand and primary input, each. The next version of this module will have greater capability of multiple types of final demands and primary inputs.

As the final note, the use of this module agrees to the following license and disclaimer:

exemplary or consequential damages (including, but not limited to, loss of use, data or profits, or business interruption) however caused and on any theory of liability, whether in contract, strict liability or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Also, we expect users of PyIO to quote this manual (as the REAL Discussion Paper) as a reference.

This manual is structured as follows. Section 2, following this introduction, outlines five things needed to get started. First will explain about the installation of Python and related extensions needed to run PyIO. Secondly, it will show how to set up PyIO. The third sub-section will discuss how input-output data are inputted into PyIO. The computation of an impact analysis will be shown as an example. Fourth section will show a typical example of a PyIO session. The last sub-section here will discuss the balance check in input-output data. Section 3 to 5 discusses the functional capabilities of PyIO. The discussion is framed within the three main PyIO functions: table operation, basic input-output analysis, and advanced input-output analysis.

2. Getting started

2.1. About Python and its installation

Python is a general-purpose open source computer programming language, optimized for quality, productivity, portability, and integration (Lutz 2001: xix.) It is easy to learn, even as the first programming language.

While we are aware that Python is frequently updated –at the moment this manual is written, version 2.3 (beta) is the latest in www.python.org— we suggest users to use Python version 2.2. We were actually informed that PyIO also works fine with the version 2.3, but no thorough test is done for that.

The installation of Python should be done in the three steps as the following:

1. Install Python version 2.2. For Windows user, you should download Python-2.2.3.exe and double click this file to install Python version 2.2.

2. Install the Windows Extension for Python version 2.2. Download Mark Hammond's Python for Windows Extensions (http://starship.python.net/crew/mhammond/) which for Python version 2.2 is named win32all-152.exe.

3. After the above two files are properly installed in your computer, you should also install the Numerical Python (Numpy), a module for numerical array computation. Windows user should install Numeric-23.0.win32-py2.2.exe. Once again, Numpy should be installed only when you have the first two files properly installed in your computer.

Having installed all of the three files, you are ready to use PyIO. You can start by double clicking `run_pyio.py`.

What hardware is required? Any Pentium-level computer should be able to run Python and PyIO. The limit is more on the software. The current version of PyIO is intended only for Windows user. We are really sorry that we cannot accommodate Mac and Linux users. All of the functions in PyIO offers an output in Microsoft Excel format. If you want this, naturally the Microsoft Excel should be properly installed in your computer. For those who do not want Excel output, PyIO will automatically write the results in an ASCII text file.

2.2. Setting up the PYIO

Once Python, the Windows extension and Numpy are properly installed in your computer, you can download the `pyio.zip` from REAL website (www.uiuc.edu/unit/real). There are four files in this zip file: `pyio.py`, `run_pyio.py`, `PyIOManual.pdf`, and `data.zip`. The first two files are needed for the PyIO session and may be put in your working directory, `PyIOManual.pdf` is the electronic version of this manual, and `data.zip` contains the data text files used in this manual. Users can copy the two files `pyio.py`, `run_pyio.py` to any directory in their computer.

You can start your PyIO session by double-clicking `run_pyio.py`. Three windows will pop up. The first is a blank window with a title `tk` on the upper left corner. In all of PyIO application, you can always ignore this window. The second window is a welcoming page of PyIO, which looks like one shown in figure 1 below. The third is a verification window. In there, you will be shown your current working directory, and asked if you want to change the current working directory. When you have all of your data files (to be explained later) in the same working directory with the two PyIO files, then there is no need to change the working directory. You may want to change the working directory if the data files are located elsewhere in your computer.

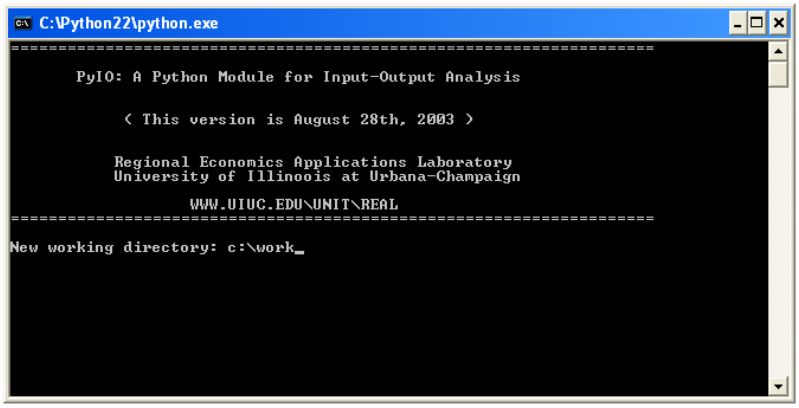


Figure 1.
Screen drop to change the working directory

If you want to change your working directory, for example to `c:\work`, then write it on the prompt as shown above. If you do not want to change the working directory, then the PyIO main menu will appear, as shown below.

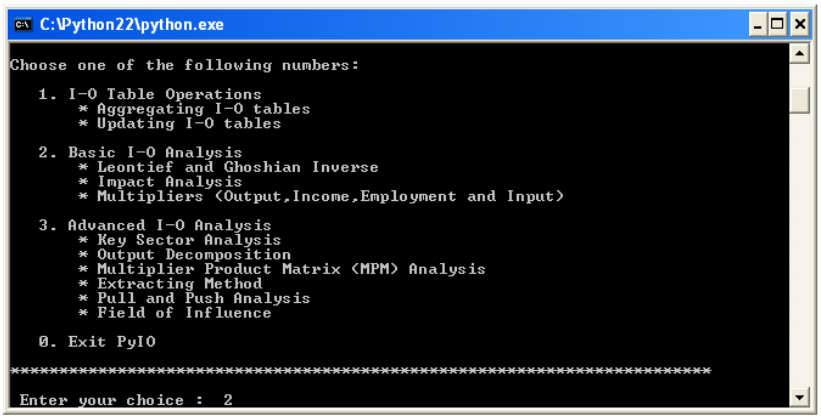


Figure 2.
Screen drop of the PyIO main menu

Now, users are ready to do input-output analysis. There are three choices for analysis and an exit. You can start the analysis by entering the number of your choice (no '2' in the above screen drop) and press enter.

2.3. Inputting an input-output dataset

Before going further with analysis, it is important to understand how a dataset is inputted in PyIO. This will be explained in this section.

For illustration, we will use the following hypothetical input-output structure. It comprises eight sectors with an aggregate final demand and value added.

Table 1. A hypothetical input-output table

Sector	1	2	3	4	5	6	7	8	Final demand	Total Output
1	16	5	24	0	6	17	10	0	622	700
2	7	17	11	48	26	0	8	0	203	320
3	43	82	33	13	17	81	51	4	283	607
4	35	9	93	7	19	99	30	2	138	432
5	19	20	19	6	59	16	16	0	220	375
6	15	15	99	45	66	11	12	7	75	345
7	25	22	47	4	42	26	45	1	349	561
8	0	0	75	0	12	7	12	3	78	187
Value added	540	150	206	309	128	88	377	170		
Total input	700	320	607	432	375	345	561	187		

Let's say that we want to put the above matrix into an input-output dataset called `datafile.txt`. This file is an ASCII file containing the input-output dataset with a particular sequence as follows:

1. First digit is the number of regions
2. Second digit is the number of sectors
3. Interindustry transaction matrix
4. Output or input values
5. Total final demand values
6. Total primary input values

The first two digits serve as identifiers of the input-output table. These identifiers play an important role in any subsequent analysis as it determines the type of matrix and, hence, its corresponding manipulations. An example of a `datafile.txt` is shown in figure 3 below.

The above input format allows one to input both national (or single-region) input-output tables as well as interregional input-output (IRIO) tables. To input a national or single region input-output table, enter '1' in the first digit (look at figure 3 below). The next set of numbers is the interindustry transactions. PyIO reads the numbers in by each line to the right. Therefore, from the `datafile.txt` below we know that z_{11} = 16, z_{12} = 5, z_{13} = 24, etc., where z is the transaction across sectors. The values of output or input, final demand, and primary input, must be inputted in the same sectoral sequence with the one in the transactions matrix.

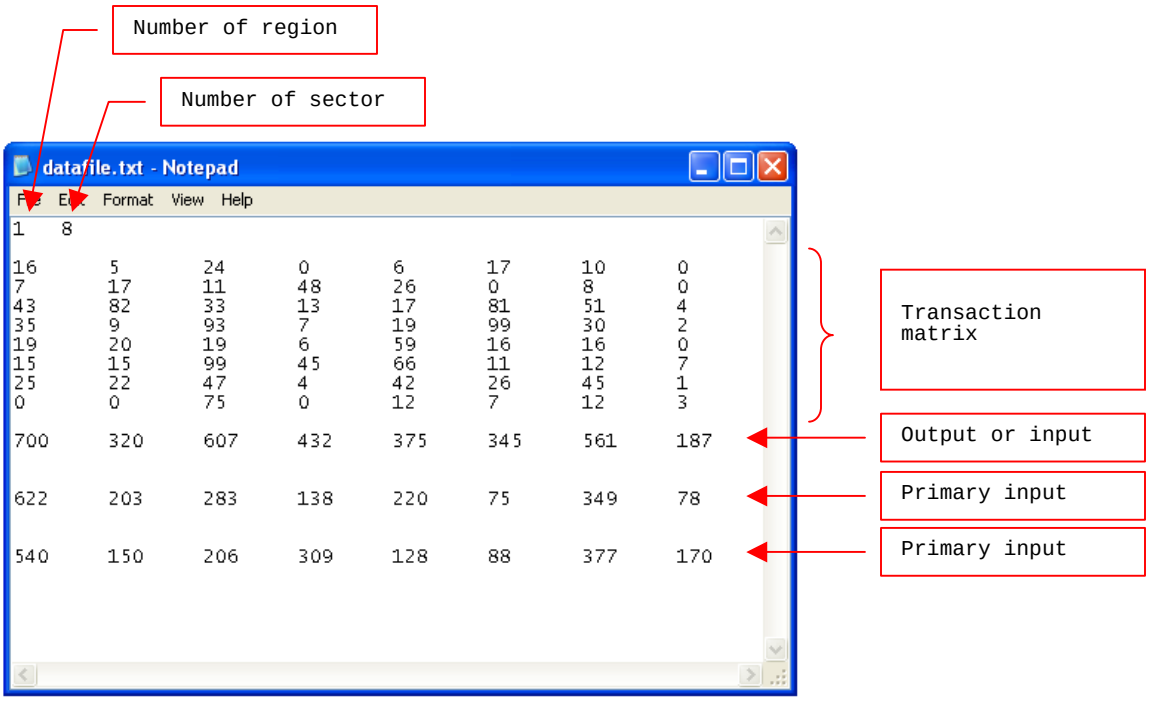


Figure 3. Structure of `datafile.txt`

To enter an interregional input-output table, put the appropriate number of regions and sectors as the identifiers. The regional-sectoral sequence is such that the sectoral index runs faster

than the regional index. If the input-output dataset shown in Figure 3 were actually from an interregional input-output with two regions and four sectors, then the first two digits would have been '2' and '4,' respectively. And, the four sectors in region 1 are presented as the first four rows/columns followed by the four sectors in region 2.

The first three items in the `datafile.txt` are required, i.e., PyIO can read a text file that only contains the two identifiers and the transaction matrix. The output, final demand, and primary input values are basically optional. However, different input-output analysis may require one or all of the last three items. For example, calculating Leontief inverse matrix requires the transaction matrix and output values. On the other hand, the extraction method requires all of the transaction, output, final demand and primary input values.

As shown in Table 1, the absence of a transaction between two sectors should be inputted as a value '0.' Do not leave a blank to denote a zero value, since the function actually counts the number of numbers inputted. For example, since the above `datafile` inputs all items, it has to have 90 numbers. If the function detects less than that, an error message will appear. If the dataset does not include primary input values, then the text file should have 82 numbers, and so on.

2.4. Running PyIO: A typical example

This part will show a typical PyIO session. For an illustration, we will do an impact analysis using the `datafile.txt` shown above, calculating the output impact of different scenarios of final demand. These scenarios should also be inputted to PyIO and they should be regarded as additional information needed to conduct the analysis. Different input-output analysis may require different additional information, which should be structured in an ASCII file in a particular structure. Regardless of the different structures, all of the additional information files should contain the first two identifiers, which must be identical to ones in the primary input-output text file (in our example here is `datafile.txt`). When the identifiers in the primary input-output file and in the additional information file are not identical, an error message will pop up.

Let's show how an impact analysis is conducted in PyIO. Assume that we have three economic scenarios that we want to check. These scenarios are stored in a text file called `sce.txt`, each line in the file represents a scenario. A more detail requirements for this scenario file will be discussed later in section 3. Here, the main purpose is to show how a session works.

Having uploaded the PyIO, users will see the main menu as shown in figure 2 above. The impact analysis is listed under the `Basic I-O Analysis`, so enter number '2' and press enter. Then, we enter the `Basic I-O Analysis` menu. The impact analysis is listed there as one of the analysis. So, enter number '3' and press enter again. Users will be asked to input the primary input-output data. Write: `datafile.txt` and press enter. Note that the file name should be inputted fully, including its `txt` extension. Then, users will be asked to input the additional information file. Thus, write: `sce.txt`. The window would seem like figure 4 below.

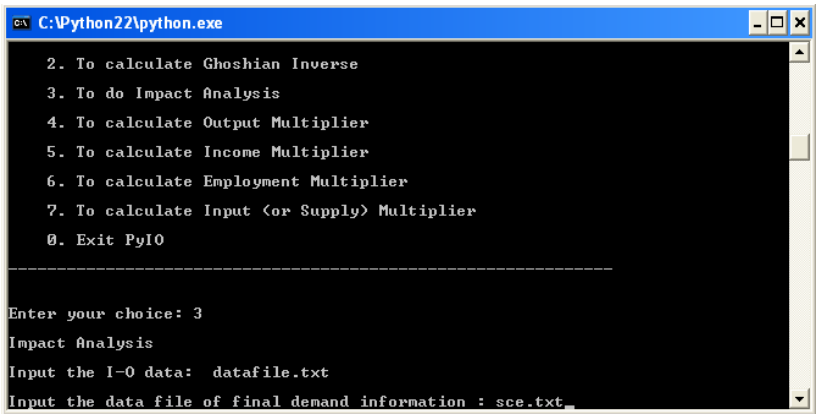
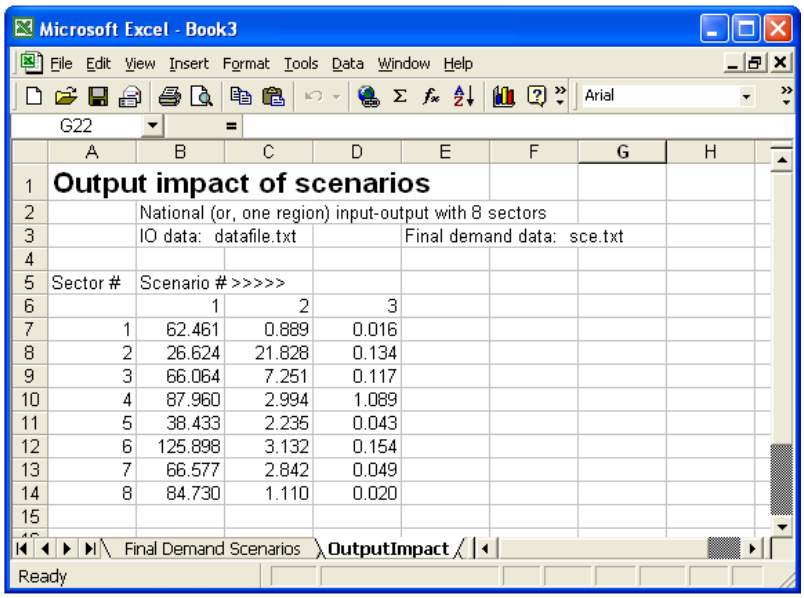


Figure 4.
Typical PyIO session

At this point, when PyIO cannot find the two files `datafile.txt` and `sce.txt` in the working directory, PyIO will close itself automatically. Users will have to upload the session by double-clicking `run_pyio.py` again.

Alternatively, when everything works fine, users will be asked whether the computation results should be printed in Microsoft Excel. If so, then the Microsoft Excel should be properly installed in the computer. For this impact analysis, the Excel output is given below (figure 5).



The Excel report typically contains the title of, and some background information about, the type of input-output (national or interregional), the number of sectors involved, and the file names of the primary input-output data and the additional information, if any.

Figure 5
Typical Excel report in PyIO

If Excel output is not preferable, the computation result will be stored in a text file. This text file will be automatically given a name, typically in accordance to the type of analysis, stored in the working directory. The information in this text file is typically similar to the one presented in Excel output.

The Python computation is fast. However, writing the results to Excel may take some significant amount of time, especially when the analysis involves an input-output of larger dimensions. For an illustration, it takes about 45 seconds to write elements of a 30x30 Leontief inverse matrix (i.e., 900 cells) in Excel using a Pentium IV 2.1 GHz. On the other hand, writing down results to a text file (i.e., saying 'no' to the question whether output should be printed in Excel), is done also very fast. Therefore, for some analysis, users may alternatively want to save the results first in a text file, and open this text file in Excel. For some very large input-output analysis, this alternative sequence may save time.

When all computation and writing results (either to Excel or to a text file) is done, a 'Well-done' window will appear, followed by another window asking whether users would like to exit the Basic I-O analysis. If no, users will stay in the Basic I-O analysis menu, and another PyIO session can be started. If users choose to exit the Basic I-O analysis, users will be checked whether they want to exit PyIO. If so, then the PyIO will stop. Otherwise, users will be brought back to the main menu (see figure 2), and another PyIO analysis can be executed.

2.5. Input-output balance check

An important part of PyIO capabilities is the input-output balance check, in particular when the input-output inputted involves the final demand and/or primary input vectors. The balance checking, in essence, evaluates whether the inputted output (i.e., the output data as written in the input-output text file) is identical to the computed output (i.e., the output data as computed using transactions, final demand and primary input vectors.) This balance check is carried out every time PyIO opens an input-output text file with final demand and/or primary input vectors.

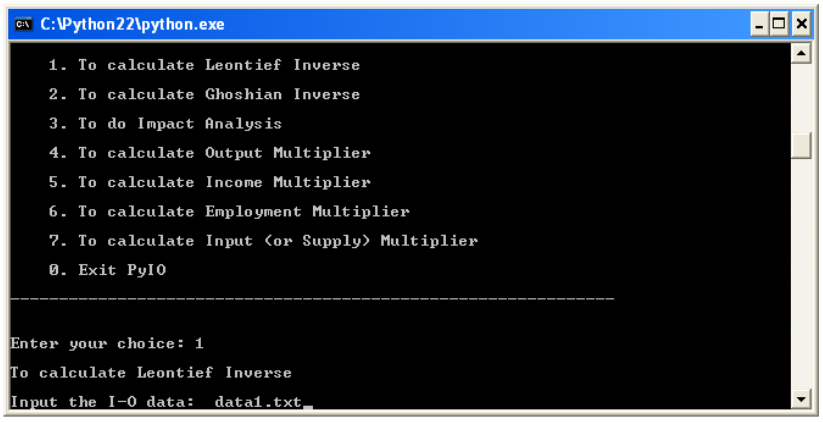
When the inputted and computed output data are identical, then PyIO will move on with the designated computation. When the two are not the same, a warning window will pop up. Users then can choose whether to continue or halt the computation. Important note: when users choose to move on with the analysis, PyIO will use the inputted (and not the computed) output in the computation. In either case, i.e., continuing with or halting the computation, PyIO will write a text file containing detail information about the nature of unbalance data.

For an illustration of this capability, let's make two changes in the correct (balanced) input-output data as shown by figure 3. These changes are:

1. The output sent from sector 1 to sector 4 is changed to 10 (from the correct one 0).
2. The output sent from sector 2 to sector 1 is changed to 7.2 (from the correct one 7).

We know exactly that this will create an unbalance input-output data since we do not change the rest of the values. Let's call this new data data1.txt, and use this file to generate the Leontief inverse matrix. This analysis is listed as the first analysis in the Basic I-O Analysis. Having entered number 2-1 from the main menu, users will be asked to input the name of input-output text file. Enter data1.txt here, and you should have a window similar to figure 6(i) below. Since the input-output data contained in data1.txt are not balance, then an error window pops up, as shown in figure 6(ii).

(i)



(ii)

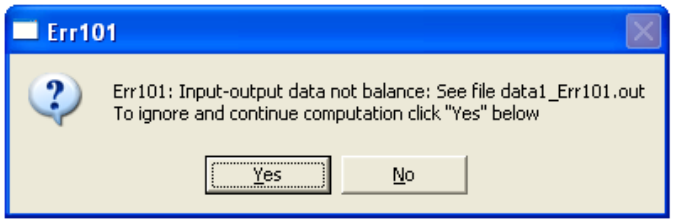


Figure 6.
(i) PyIO session for Leontief inverse;
(ii) Unbalance data prompt

Regardless whether users want to stop or continue with computation, a text file containing the detail of the error is created and named after the primary input-output data. In this example, this error information is automatically named `data1_Err101.out`. Clicking 'Yes' in the warning window will get PyIO to continue the computation of the Leontief inverse matrix, and users can see the error file `data1_Err101.out` later. Clicking 'No' will stop the computation of the Leontief inverse matrix.

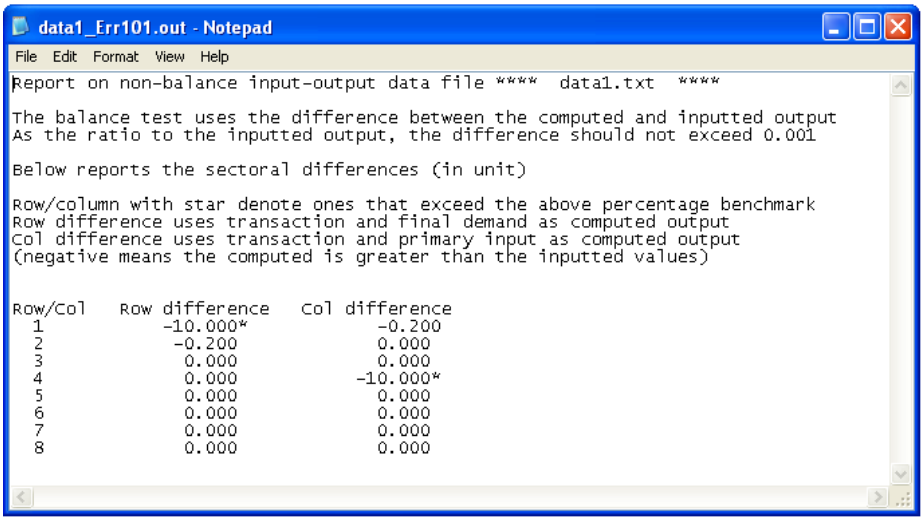


Figure 7
Contents of error information file

The error report starts with the title and the name of input-output text file being examined. The unbalance criterion deserves some attention here. PyIO acknowledges that the criterion should not be based on the level of differences between the inputted and computed outputs, because one input-output table may have entries in the vicinity in thousands, others may have ones in the vicinity of millions. Thus, a difference in a magnitude of '1' can mean one thousand or one million, depending on the input-output table at hand.

Therefore, PyIO chooses different criterion: the ratio of the differences (between the inputted and computed outputs) to the total output, for each sector. There is still another problem. Some sectors may be markedly different in magnitude from others. In the real input-output, some sectors may have output at a 7-digit magnitude while others, because they are not so connected to the rest of the economy, may be only at 3-digit magnitude. In order to demote this problem, PyIO takes the average number of (output) digits from each of the sectors, denote this as k, and uses 10^{-k} as the benchmark. That is, the ratio between the difference (between the inputted and computed outputs) and the total output should not be greater than 10^{-k}. In the above example, all of the outputs are at 3-digit magnitude. Thus, the criterion is that the difference, as the ratio to the actual output, should not be more than 10^{-3}, or 0.001.

The actual reporting of differences, by sector, follows. Note that the error file reports the actual differences, and not the ratio. The difference is positive (negative) when the inputted is greater (smaller) than the computed values. An asterisk is put to the right of the difference (either column or row), when it actually exceeds the benchmark. In the above example, the first change results in a difference greater than the benchmark, while the second change does not.

3. Table operations

3.1. Aggregation of input-output tables

The aggregation of input-output tables is a common procedure in input-output analysis. In the case of a national table (or one region table), aggregation of input-output tables can be conducted between sectors, while in the case of inter-regional input-output both sectoral and regional aggregations can be considered.

Suppose we want to aggregate an *n*-sector input-output table into a *k*-sector specification, where *n* is greater than *k.* A crucial element in the aggregation procedure is the construction of an aggregation matrix $\mathbf{S}$ that is of $k \times n$ dimension. For example, to aggregate sector 2 and sector 3 in a 4-sector input-output table, the aggregation matrix will be:

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1)$$

The new transaction matrix, total output, final demand for the aggregated table can be accomplished by multiplying the aggregation matrix $\mathbf{S}$ with the original transactions matrix. Let

NT and **T** be the new transaction matrix and old transaction matrix respectively, **NX** and **X** be the new and old total output, **NF** and **F** be the new and old final demand.

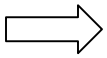
$$\begin{aligned} \mathbf{NT} &= \mathbf{STS}' \\ \mathbf{NX} &= \mathbf{SX} \\ \mathbf{NF} &= \mathbf{SF} \end{aligned} \qquad (2)$$

Note that to aggregate sectors or regions in the inter-regional input-output table a similar procedure is followed to sectoral aggregation scheme except for the different structure of the aggregation matrix.

The table aggregation requires a file containing the aggregation information, i.e., how sectors are aggregated together. Note that each line in this aggregation info file will form a new sector. Let's show an example when we want to aggregate the input-output table shown in table 1, from an 8-sector to a 4-sector classification. Suppose that we name the aggregation info file as `agg_info.txt`. Figure 8 below shows a typical aggregation criterion and its corresponding `agg_info.txt`. Note that the aggregation info text file does not require the usual two identifiers about the number of regions and number sectors.

A valid `agg_info.txt` has several requirements. First, each of the old sectors must only be used once in the new sector structure. Second, the `agg_info.txt` file should have no blank lines. PyIO will return an error in either case.

New sector	Old sector
1	1,6
2	2,3,4
3	5,7
4	8



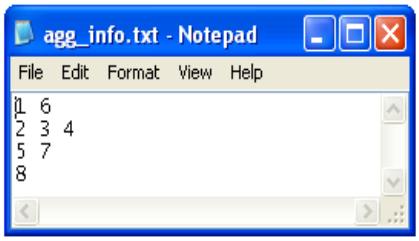


Figure 8. Aggregation and `agg_info.txt`

When prompted by PyIO, users should input the file names of the primary input-output text file and the aggregation information file. Results can be written either in Microsoft Excel or text file. In Excel file, there will be two worksheets: one named "aggregated table" describes the aggregated table, the other one named "agg_info" shows that corresponding aggregation information between the new table and the old one. The text file of the results contains the same information as the Excel file, with a name of "`agg_`" being a prefix, "`.out`" being the suffix of data file.

PyIO automatically creates a new input-output text file for the newly aggregated table. Since the primary input-output is named `datafile.txt`, the aggregated input-output text file is named `datafile_new.txt`. The contents of this new text file depend on the contents of the primary input-output table. If the primary file contains output, final demand and primary input, then the new text file will also do so.

Finally, aggregation not only can be done in terms of sectors, but also in terms of regions. Usually this procedure is applied to an interregional input-output table. The aggregation information file will then pertain to the number of regions, rather than number of sectors. In the regional (or spatial) aggregation, the composition of sectors is intact. Aggregating an interregional input-output table in terms of sectors and regions can thus be done in two steps.

3.2. Updating an input-output table

Updating techniques are widely used in input-output analysis to overcome the expensive cost of producing new tables from surveys. Updating an input-output can be done using either non-survey or partial-survey methods. At this point, the program includes two functions to update an input-output table using non-survey methods: one employs the location quotient, and the other uses the regional supply percentage. These two functions will be elaborated in this section. A partial-survey method, known as the bi-proportional or RAS method, is elaborated in the subsequent section.

The regional supply percentage and location quotient are commonly used to regionalize a national input-output table. This technique is frequently used when the only available input-output data is the national table supplemented by regional data, such as gross output or employment by sector.

The regional supply percentage in region R in each sector *i* (p_i^R) can be computed as the following:

$$p_i^R = \frac{X_i^R - E_i^R}{X_i^R - E_i^R + M_i^R} \tag{3}$$

where *X*, *E* and *M* respectively denote output, export and import. The magnitude of the regional supply percentage lies between 0 and 1, with the higher end corresponding to regional self-sufficiency in sector i. To adjust the national table, each row of the national $\mathbf{A}$ matrix will be multiplied by the appropriate percentage. In matrix notation this is written as $\mathbf{A}^{\mathbf{RR}} = \hat{\mathbf{P}}\mathbf{A}$, where $\mathbf{A}^{\mathbf{RR}}$ is the regional input coefficient matrix, $\hat{\mathbf{P}}$ is the *n*-element vector of regional supply percentage, and $\mathbf{A}$ is the national input coefficient matrix.

The location quotient also measures some degree of self-sufficiency at the regional level. Here we will use the simple location quotient, which for region R and sector i is defined as

$$SLQ_i^R = \frac{X_i^R / \Sigma_i X_i^R}{X_i^N / \Sigma_i X_i^N} \tag{4}$$

where the superscript R and N, respectively, denote regional and national data. The updating criteria are given as the following:

$$a_{ij}^{RR} = \begin{cases} a_{ij}^{N}(SLQ_i^R) & \text{if} \quad SLQ_i^R < 1 \\ a_{ij}^{N} & \text{if} \quad SLQ_i^R \geq 1 \end{cases} \tag{5}$$

The additional information required to compute the regional supply percentage (and also the simple location quotient) should be inputted in a separate ASCII text file. This file, as usual, should contain identical identifiers with the primary input-output data file.

Suppose that we want to update the input-output table as shown `datafile.txt`. Let's say that `datafile.txt` is a national input-output table, and we want to generate a regional input-output for region R using the regional supply percentage method. Thus, the information to calculate the RSP for region R should be inputted in another text file, let's name it `region_R.txt`. This file should contain the values of exports, imports, and output (exactly in that sequence) for each sector in the economy. Thus, this file should contain exactly 2+3*d* numbers, where *d* is the number of sectors.

Differently, suppose we want to update `datafile.txt` to generate a regional input-output table for region S using the simple location quotient method. The information needed is the national and regional dataset to compute the location quotient. Again, this information should be inputted in another text file, for example let's name it `region_S.txt`. In addition to the usual identifiers, this `region_S.txt` should contain values of national and regional sectoral variables (exactly in that sequence). This file should only contain 2+2*d* numbers, where *d* is the number of sectors. Note that users can use any kind of national-regional dataset to compute the location quotient. Among those variables that are of common interest are output, employment or income.

Having inputted the appropriate file names of the primary input-output data and the updating information file, users will be asked whether they want to create a new datafile based on the new regional input coefficient matrix. This datafile may be used for subsequent input-output analysis in PyIO. The new datafile will be a text file (`.txt`) with the same name as the earlier updating info file plus a suffix '`_RSP`' when the updating uses regional supply percentage method, and '`_SLQ`' when the updating uses the simple location quotient. In our example above, the new data file for region R will be named `region_R_RSP.txt`, and that for region S would be `region_S_SLQ.txt`. This new data file will only contain the usual identifiers, transaction matrix and output. To use this new datafile in PyIO analysis that requires more data (i.e., final demand and primary input), users need to enter them manually using a text editor.

To create a new data file using SLQ method, users need to input another text file containing the regional output. This text file is needed, because using SLQ, no information is given about the extent of regional output. This file, as usual, needs to contain the usual two identifiers and the sectoral regional output. This file should contain 2+*d* numbers, where *d* is the number of sectors. This regional output text file is not needed for the RSP method. The reason is because the regional output data has been earlier inputted in the updating information text file. PyIO takes the regional output data directly from this file. The example for this regional output text file can be seen in file `output_S.txt`.

As usual, the new regional input coefficient matrix can be printed in Microsoft Excel. If users do not want the output in Microsoft Excel format, an output text file will be created. It will be named

using the `datafile` name, plus a suffix `_SLQUpdate.out` or `_RSPUpdate.out`. In the output text file, only the newly updated $\mathbf{A}$ matrix is presented.

3.3. RAS method

A widely used partial survey method to update an input-output table is the bi-proportional or RAS method. This method is commonly used to update an input-output matrix over a short period of time as well as to adjust a national table in order to derive a regional matrix. A lengthy exposition of this technique can be found in Chapter 8 of Miller and Blair (1985).

The method finds a new input coefficient matrix $\mathbf{A}$ in, for period t given the prior $\mathbf{A}$ matrix in period 0, and some additional information for period *t*. The period *t* information needed are sectoral allocation of intermediate output (*U*), sectoral allocation of intermediate input (*V*) and sectoral output (*X*). The general structure of the RAS method is

$$\mathbf{A}^{2n} = [\mathbf{R}^n \ldots \mathbf{R}^1]\mathbf{A}(0)[\mathbf{S}^1 \ldots \mathbf{S}^n] \tag{6}$$

where $\mathbf{A}(0)$ is the prior matrix to be adjusted using series of adjustments coefficients represented in $[\mathbf{R}^n \ldots \mathbf{R}^1]$ and $[\mathbf{S}^1 \ldots \mathbf{S}^n]$. In the actual computation, the method involves a series of iterations up to the point where the absolute difference between the row (and column) sums of transaction (i.e., $\mathbf{AX}$) matrix and the values of *U* (and *V*) are less than 0.001. This convergence criterion should be good enough for any reasonable application of RAS method. In the case that the iterations do not get to satisfy the convergence criteria, PyIO will stop at the 50th iteration, and the $\mathbf{A}$ matrix obtained at that step will be used in the reporting.

In PyIO application, the RAS method actually updates the input coefficient $\mathbf{A}$ matrix. In several applications, the user may wish to update the transactions matrix rather the input coefficient. This can be easily done by inputting the transaction matrix in the input-output data file, and put values of 1 as output values.

The updating information –vectors *U*, *V* and *X* – should be inputted in a separate text file. The structure of this ASCII text file follows the standard format with two usual identifiers in the beginning. The rest must be sequenced as follows: sectoral allocation of intermediate output (*U*), sectoral allocation of intermediate input (*V*) and sectoral output (*X*). Let's suppose this file is named `time_t.txt`, as shown below.

Users will be prompted to decide whether to specify a new datafile based on the updated regional input coefficient matrix. This new datafile may further be used for subsequent computation of Leontief inverse, impact analysis or other analysis in PyIO. To use this new datafile in PyIO analysis that requires more data (i.e., final demand and primary input), users need to enter them manually using a text editor.

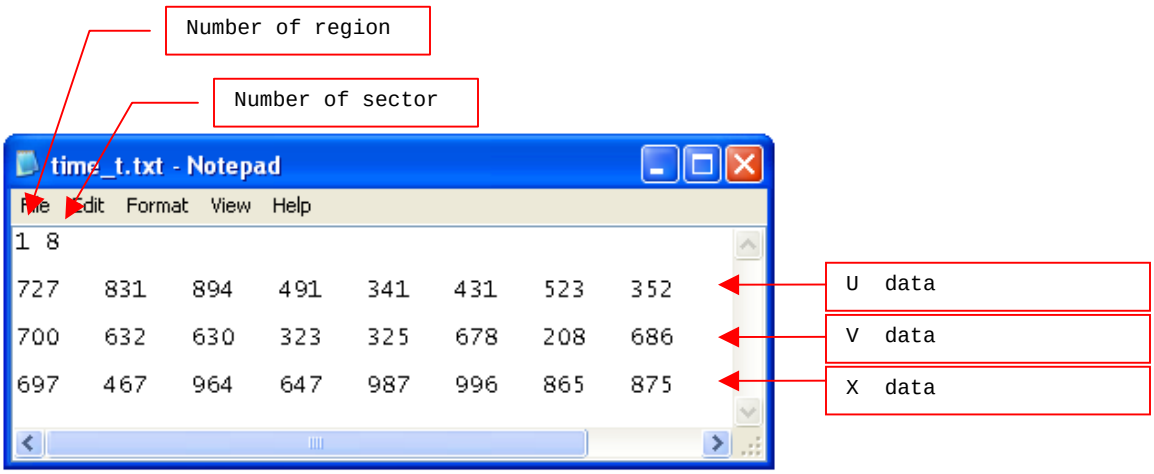


Figure 9. Structure of `time_t.txt`

The new data file will be a text file (`.txt`) with the same name as the earlier input-output data file, plus a suffix '_RAS'. This new data file only contains the usual identifiers, transaction and output. To use this new datafile in functions that requires more data (i.e., final demand and primary input), users need to put them manually using a text editor.

As usual, the new regional input coefficient matrix can be printed in Microsoft Excel. Otherwise, a text file will be created. Given the file names above, this output text file will be named `time_t_RASUpdate.out`.

4. Basic input-output analysis

4.1. Leontief and Ghoshian inverse matrices

The generation of multipliers from input-output models may be considered as one of the most popular uses of the modeling system. Impact analysis (illustrated in section 4.2) provides insights into the way in which changes in one sector of an economy affect all other sectors; the magnitude of these impacts, the strengths of the linkages between sectors, the size and complexity of the economy all combine to produce a range of multiplier values. In this section, multipliers generated from the Leontief and Ghoshian inverses will be presented; in addition to their use in impact analysis, these multipliers also appear as part of key sector analysis introduced in section 7.

The Leontief inverse is practically the most widely derived matrix from an input-output table. While the Leontief inverse is based on the input requirement matrix $\mathbf{A} = [a_{ij}]$, the Ghoshian inverse is based on the output allocation matrix $\vec{\mathbf{A}} = [\vec{a}_{ij}]$. The elements of these matrices are computed in the following way:

$$a_{ij} = \frac{z_{ij}}{X_j} \qquad \text{and} \qquad \vec{a}_{ij} = \frac{z_{ij}}{X_i} \tag{7}$$

where z_{ij} is the input from i required in the production of j (or in Ghoshian structure is the output of i sent to the production to j), and X is the total input or output. In practical terms, the input requirement matrix is obtained by dividing each cell in the transaction matrix with its corresponding input in each column; while the output allocation matrix is obtained by dividing each cell in the transaction matrix with its corresponding output in each row. The Leontief inverse matrix is then computed as $(\mathbf{I}-\mathbf{A})^{-1}$, while the Ghoshian inverse is then obtained by $(\mathbf{I}-\vec{\mathbf{A}})^{-1}$.

In PyIO, the computation of these inverses requires an input-output text file that contains at the minimum the transaction and output data. Users will also be prompted to indicate whether Microsoft Excel format is preferred. If so, a Microsoft Excel worksheet will be opened, and the Leontief or Ghoshian inverse matrix will be written. If users do not want the Microsoft Excel format, the output will be presented in an ASCII text format, placed in the same working directory. For our `datafile.txt` the output text file of Leontief inverse will be named `datafile_LeontInv.out` and that of Ghoshian inverse will be `datafile_GhoshInv.out`.

The maximum number of sectors (or combination of region-sector) that an input-output dataset can have is limited by the fact that an Excel worksheet only has 256 columns. If you are dealing with an input-output dataset with greater number of dimensions, do not send the output to Python when prompted by the function. Instead, send it to an output text file, and you can easily open this file in Excel spreadsheet.

4.2. Impact analysis

Impact analysis is one area where input-output is widely used; applications here typically require the analyst to prepare an estimate of the impact of a change in one or more sectors on the economy as a whole or on individual sectors. The analysis computes a new set of output if the economy is given new hypothetical values of final demand. In particular, the usual formula is $\mathbf{X} = (\mathbf{I}-\mathbf{A})^{-1}\mathbf{F}$ where $\mathbf{X}$ is output, $(\mathbf{I}-\mathbf{A})^{-1}$ is the Leontief inverse, and $\mathbf{F}$ is final demand.

The impact analysis requires the scenarios to be inputted in a separate text file. This text file should contain the two identifiers and final demand scenario(s). Earlier in section 2.3 we have seen typical scenarios inputted in a file named `sce.txt`. As shown, PyIO is capable of handling multiple scenarios of final demands. Practically, there is no limitation on the number of scenarios, for which output impact, that PyIO can compute at one shot. More limitation is put by the Excel spreadsheet, because the output impact of each scenario is reported in Excel column. Since an Excel worksheet only contains 256 columns, then this is the maximum number of scenario that `sce.txt` can contain, if users want the output in Excel file. If output will be later stored in an output text file, this limitation does not apply.

As before, the code will read `sce.txt` file by line to the right. Therefore, it does not really matter if the scenarios are stacked to the right, instead of line by line. What really matters is the fact that the sectoral sequence of each scenario must conform to one in the `datafile.txt`. As before, the final demand scenario in the interregional input-output framework is easily constructed using the two identifiers in the beginning of this scenario text file. The regional-sectoral sequence of the scenarios must also conform to the regional-sectoral sequence in the interregional transaction matrix.

If one chooses to have the output in Microsoft Excel format, the PyIO will open a Microsoft Excel file. Two worksheets will be written: 'Final Demand Scenarios' and 'Output Impact.' The 'Final Demand Scenarios' worksheet will rewrite the scenarios given in `sce.txt`, and the 'Output Impact' worksheet will produce the corresponding output impact. Each final demand structure and its corresponding impact are arranged in each column.

If users choose not to have the output in Microsoft Excel format, then an output text file is automatically created. Given that the scenarios above is contained in `sce.txt`, then the output text file is written to a file named `sce_Impact.out`.

4.3. Multiplier Analysis

Following Miller and Blair (1985), there are four types of multipliers that are possible to calculate from an input-output matrix: output multiplier, income multiplier and employment multiplier. The output multiplier is calculated as the column sum of the Leontief inverse. Given that $\mathbf{B} = [b_{ij}]$ as the Leontief inverse matrix, then the output multiplier for sector j, denoted by O_j, is given by

$$O_j = \sum_{i=1}^{n} b_{ij} \tag{8}$$

where n is the number of sectors. The computation of income multiplier requires the use of wage vector (in the primary input table) to calculate the household input coefficient. This coefficient makes up the $(n+1)^{th}$ (household) row that is used in closing the model with respect to households, indicating household income received per dollar's worth of sectoral output (Miller and Blair, 1985: 105.) Denoting this coefficient with $a_{n+1,i}$, it is defined as $a_{n+1,i} = z_{n+1,i} / X_i$ where z is transaction and X is output. Given the Leontief inverse $\mathbf{B}$, the household income multiplier is formulated as

$$H_j = \sum_{i=1}^{n} a_{n+1,i} b_{ij} \tag{9}$$

The employment multiplier would require the use of sectoral employment to calculate the labor input coefficient. This coefficient $w_{n+1,i}$ is given as $w_{n+1,i} = e_i / X_i$, where e is the employment. The employment multiplier for sector j is then given by the following formula

$$E_j = \sum_{i=1}^{n} w_{n+1,i} b_{ij} \tag{10}$$

Finally, the input (or supply) multiplier is computed from the Ghoshian inverse. The actual computation is identical to the output multiplier, given a Leontief inverse matrix. Therefore, if $\vec{\mathbf{B}} = [\vec{b}_{ij}] = (\mathbf{I} - \vec{\mathbf{A}})^{-1}$, then the input (or supply) multiplier is given as

$$O_j = \sum_{i=1}^{n} \vec{b}_{ij} \tag{11}$$

As should have been clear, the computation of input or output multiplier does not require any information other than the primary input-output text file. However, the computation of income, or employment multiplier does require additional information. Income multiplier requires data on income, which are typically inputted as parts of the primary input matrix. This information should be inputted in a separate text file, that contains the usual two identifiers and the sectoral total income values. In the input-output application, income is typically generated by the wages and salaries. More broadly, one may also assume that income comprises wages, salaries, operating surplus (profit) and interest payment. In any case, the income data inputted in this information file to compute income multiplier should be the total income. This file should only contain 2+*d* numbers, where *d* is the number of sectors in the input-output data. An example of this file can be seen in `income.txt`.

On the other hand, the computation of employment multiplier requires additional text file containing the number of employment by sectors. This data is not a part of the input-output table, so they should be elsewhere found. Again, this file only contains 2+*d* numbers, where *d* is the number of sectors in the input-output data. An example of this file can be seen in `employ.txt`.

As usual, users will be asked whether output in Microsoft Excel is preferred. The output in Microsoft Excel will also contain the type of input-output model being employed (national or interregional) with its corresponding dimensions. The reporting format in Excel for each multiplier allows for a very large dimension of input-output. At this moment, the maximum number of Excel rows is more than 65,500 lines, which practically puts no limit on the input-output dimension for the multiplier analysis.

If Microsoft Excel is not preferred, an output text file will be automatically created. The name of this output text file will be taken from the name of primary input-output text file, plus a suffix `_[type]Mult.out` where [type] can be `Out` for Output, `Inc` for Income, `Emp` for Employment multiplier, and `Inp` for Input multiplier. For example, the output multiplier of the earlier `datafile.txt` will be written to a text file named as `datafile_OutMult.out`.

5. Advanced input-output analysis

5.1. Key-sector Analysis

Key sector analysis is widely used in input-output analysis. It aims to identify those sectors whose economic activity exerts a greater than average influence on the whole economy. Policy makers interested in industrial targeting, government officials considering incentives for new industry, or school officials anxious to estimate the fiscal impact of the closure of a major firm or government installation are all users of key sector analysis. In essence, the search for key sectors is based on an assumption that some activities in an economy have the potential to generate greater growth and through their backward and forward linkages, spur the growth of the rest of the economy. The idea has always been controversial; many authors have claimed that the methods used to identify key sectors are based on ex post relationships and provide no indication of the future prospects for these sectors. Further, the idea that all sectors are interdependent and yet a subset of these are key sectors may appear to be contradictory; the non key sectors may be the "glue" that holds the economy together. Two other methods are closely associated with key sector analysis – the extraction method (described in section 12) and cluster analysis. The methodology for cluster analysis will be included in a future volume of the software.

In this section, key sectors are identified by calculating backward and forward linkage proposed by Rasmussen (1956) drawing on entries in the Leontief inverse. Let $\mathbf{B}=(\mathbf{I}-\mathbf{A})^{-1}=[b_{ij}]$ be the Leontief inverse matrix and let $B_{\cdot j}$ and $B_{i\cdot}$ be the column and row multipliers of this Leontief inverse. The sector *j*'s backward linkage (BL_j) and forward linkage (FL_i) are defined as:

$$BL_j=\frac{\frac{1}{n}\sum_{i=1}^{n}b_{ij}}{\frac{1}{n^2}\sum_{i,j=1}^{n}b_{ij}}=\frac{\frac{1}{n}B_{\cdot j}}{\frac{1}{n^2}V}=\frac{B_{\cdot j}}{\frac{1}{n}V} \qquad \text{and} \qquad FL_i=\frac{\frac{1}{n}\sum_{j=1}^{n}b_{ij}}{\frac{1}{n^2}\sum_{i,j=1}^{n}b_{ij}}=\frac{\frac{1}{n}B_{i\cdot}}{\frac{1}{n^2}V}=\frac{B_{i\cdot}}{\frac{1}{n}V} \tag{12}$$

where, $B_{\cdot j}=\sum_{i=1}^{n}b_{ij}$, $B_{i\cdot}=\sum_{j=1}^{n}b_{ij}$ and $V=\sum_{i=1}^{n}\sum_{j=1}^{n}b_{ij}$.

The usual interpretation is to propose that, if BL_j >1, a unit change in final demand in sector j will generate an above-average increase in activity in the economy. Similarly, for FL_i >1, it is asserted that a unit change in all sectors' final demand would create an above average increase in sector i. Thus, a key sector is identified as one having both indices greater than 1.

As usual users will be prompted whether results in Microsoft Excel is preferable. If so, a Microsoft Excel file with two worksheets will be opened. The first worksheet, named `Key Sectors`, shows the unsorted backward and forward linkages. The second worksheet, named Linkages indicators, presents the sorted backward and forward linkages. If output in Microsoft Excel is not preferable, then the output will be written in an output text file named after the

primary input-output text file. In our example using the `datafile.txt`, the output file for the key-sector analysis will be `datafile_KeySector.out`.

5.2. Decomposition of output changes

The decomposition technique analyzes differences in two-period sectoral output into three different parts. The technique is based on Sonis et al. (1996), where they note that the output change $\Delta\mathbf{X}$ can be stated in the following way:

$$\begin{aligned}\Delta\mathbf{X} &= \mathbf{B}_t\mathbf{f}_t - \mathbf{B}_0\mathbf{f}_0 \\ &= (\mathbf{B}_0 + \Delta\mathbf{B})(\mathbf{f}_0 + \Delta\mathbf{f}) - \mathbf{B}_0\mathbf{f}_0 \\ &= \mathbf{B}_0\Delta\mathbf{f} + \Delta\mathbf{B}\mathbf{f}_0 + \Delta\mathbf{B}\Delta\mathbf{f} \\ &= \Delta\mathbf{X}^{\mathbf{f}} + \Delta\mathbf{X}^{\mathbf{B}} + \Delta\mathbf{X}^{\mathbf{Bf}}\end{aligned} \tag{13}$$

where $\mathbf{X}$ is output, B is the usual Leontief inverse, and $\mathbf{f}$ is final demand, subscripts '0' and 't' denotes two time periods. The decomposition results in three different components. The first component ($\Delta\mathbf{X}^{\mathbf{f}}$) is the part of output change that is due to changes in final demand. The second component ($\Delta\mathbf{X}^{\mathbf{B}}$) pertains to the output change that is due to technological progress (i.e., due to changes in the Leontief inverse matrices), and the last part ($\Delta\mathbf{X}^{\mathbf{Bf}}$) is the part due to synergistic interaction between final demand and technological change.

Further, the decomposition can be made to trace the output changes by determining whether they originated from the sector itself or from other sectors in the economy. The two components are referred to as *self-generated* and *non-self-generated* changes, respectively. For a certain sector *i*, *self-generated* changes can be obtained by using b_{ii}, f_i, and their changes through time. If $\mathbf{s\Delta X}$ represents self-change of output, $\mathbf{ns\Delta X}$ represents non-self-change of output, the self-change and non-self-change of each part in will be:

$$\mathbf{s\Delta X_i^f} = b_{ii}\mathbf{\Delta f_i} \quad ; \quad \mathbf{ns\Delta X_i^f} = \mathbf{\Delta X_i^f} - \mathbf{s\Delta X_i^f}$$

$$\mathbf{s\Delta X_i^B} = \Delta b_{ii}\mathbf{f_i} \quad ; \quad \mathbf{ns\Delta X_i^B} = \mathbf{\Delta X_i^B} - \mathbf{s\Delta X_i^B}$$

$$\mathbf{s\Delta X_i^{Bf}} = \Delta b_{ii}\mathbf{\Delta f_i} \quad ; \quad \mathbf{ns\Delta X_i^{Bf}} = \mathbf{\Delta X_i^{Bf}} - \mathbf{s\Delta X_i^{Bf}}$$

For the PyIO session, the two IO data files, with identical identifiers, should have the minimum information of transactions, output and the final demand. After the decomposition is preformed, the user will again be prompted to choose whether output in Microsoft Excel format is preferred. If users do not want the output in Excel then the output will be automatically written in an output text file at time-t with suffix "`_Decomp.out`." In these reports, nine parts of the above decomposition are reported: $\Delta\mathbf{X}^{\mathbf{f}}, \Delta\mathbf{X}^{\mathbf{B}}, \Delta\mathbf{X}^{\mathbf{Bf}}$ are denoted as Decom_1, Decom_2 and Decom_3; $\mathbf{s\Delta X_i^f}, \mathbf{s\Delta X_i^B}, \mathbf{s\Delta X_i^{Bf}}$ are denoted as SelfCh_1, SelfCh_2, and SelfCh_3, and

$\mathbf{ns\Delta X_i^f}, \mathbf{ns\Delta X_i^B}, \mathbf{ns\Delta X_i^{Bf}}$ are denoted as NonSelfCh_1, NonSelfCh_2, and NonSelfCh_3, respectively.

5.3. MPM Analysis

The structure of economies change over time; users interested in providing a visual appreciation of structural changes can easily accomplish this with use of a simple method to be presented here. If the interest is in comparing the structure of two economies at the same point in time, then the same methodology will be of value. The input-output multiplier product matrix (MPM) is a visualization technique derived from the Leontief inverse matrix. The MPM matrix, **M**, is defined as (see Sonis et al. 1997):

$$\mathbf{M} = \left[m_{ij} \right] = \frac{1}{V} \left[B_{i\cdot} B_{\cdot j} \right] = \frac{1}{V} \begin{pmatrix} B_{1\cdot} \\ B_{2\cdot} \\ \vdots \\ B_{n\cdot} \end{pmatrix} \begin{pmatrix} B_{\cdot 1} & B_{\cdot 2} & \cdots & B_{\cdot n} \end{pmatrix} \tag{14}$$

where $\mathbf{B} = [b_{ij}]$ is the Leontief inverse, V is the grand sum of rows and columns of **B** or $V = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij}$, $B_{i\bullet} = \sum_{j=1}^{n} b_{ij}$ and $B_{\bullet j} = \sum_{i=1}^{n} b_{ij}$. Note that the column and row multipliers from MPM are the same as those from the Leontief inverse matrix:

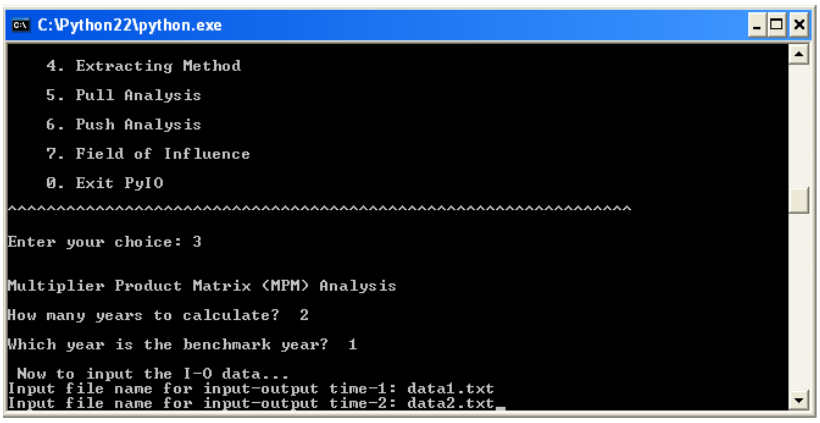
$$\sum_{j=1}^{n} m_{ij} = \frac{1}{V} \sum_{j=1}^{n} B_{i\cdot} B_{\cdot j} = B_{i\cdot} \qquad \text{and} \qquad \sum_{i=1}^{n} m_{ij} = \frac{1}{V} \sum_{i=1}^{n} B_{i\cdot} B_{\cdot j} = B_{\cdot j} \tag{15}$$

Thus, the MPM structure is essentially connected with the properties of sector backward and forward linkages. The rows and columns of matrix M can be rearranged along the magnitude of the values of backward and forward from the largest to the smallest to provide the hierarchy of backward (for columns) and forward (for rows) linkages. Using the MPM matrix, it is possible to construct *economic landscapes* to provide a summary view of the economic structure.

One use of the MPM is to trace the structural change over time. Sonis et al. (1997) suggested to ”freeze” the ordering of rows and columns at one point in time, and to use this ordering to compare the economic landscapes from different time periods. The analysis highlights the extent to which the structure has remained stable or changed over time. If the hierarchy is the same, the structure has not changed; disruptions to the hierarchy at subsequent periods provide a sense of the magnitude of the changes. Detailed sector-by-sector analysis can thus focus on the sector(s) in which the dominating changes have been observed.

When the MPM Analysis is chosen (no. 3 in the `Advanced I-O Analysis`), users will be prompted to several questions. First asks the number of years involved in the analysis. Second asks the year which is going to be used as the benchmark year. Note that in PyIO the numbering of years starts with 1 (not 0). For example, if users wants to do an MPM analysis

involving two-period input-output and uses the first year as the benchmark, then they will answer the first question with '2' and the second question with '1.' Following those two questions, user will then asked to input the names of input-output text files for the MPM analysis.



See figure 10 on the left. Naturally, each period input-output should be inputted in a text file. In the example here, suppose that the two input-output text files are `data1.txt` & `data2.txt`.

Figure 10. Screen drop of MPM Analysis in PyIO

The results will be written either as Excel or text file upon users' preferences. In Excel, the number of worksheets will be (2 × number of years + 1). For our example, since the analysis involves two years of input-output, then there are 5 worksheets written. See the screen drop below (figure 11). The first two are named as "`MPM-1`" and "`MPM-2`" containing MPM results written in the sector order of its own time. See that the sectors are ranked, and these ranks are different for time-1 and time-2. The next two worksheets named "`MPM (ordered)-1`" and "`MPM (ordered)-2`" show the MPM results in the sector order of time-1 (i.e., the benchmark year). The last worksheet is "`Datafiles`" listing two files corresponding to time-1 and time-2.

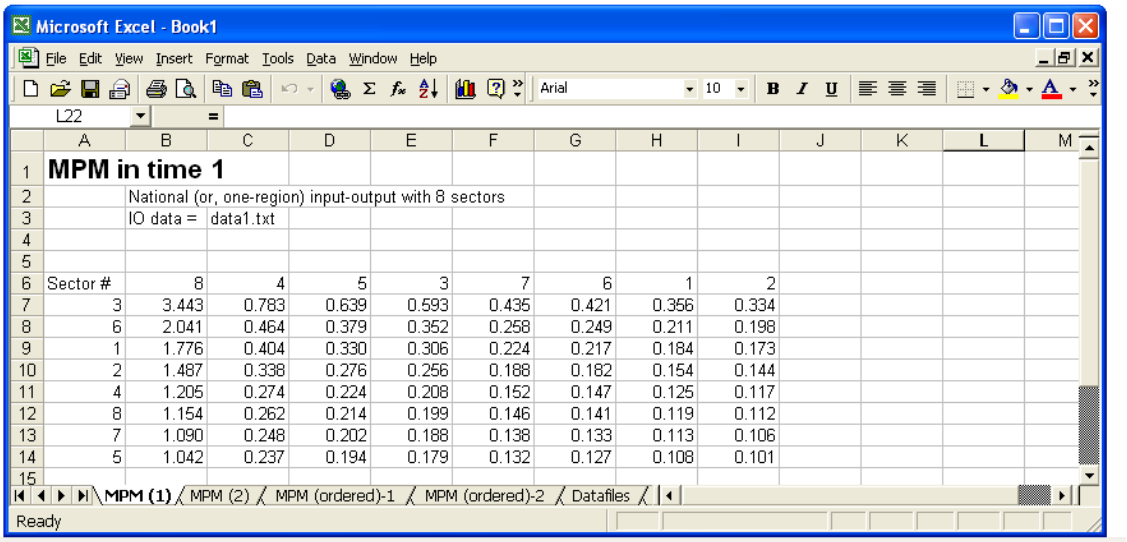


Figure 11. Screen drop of Excel results of MPM

If results written in text file are preferred, there will be a warning message to tell users that the results will be saved as a text file in the working directory. Given the above file names, this output text file will have name `data2_MPM(2,1).out`. The numbers in parentheses pertain to the number of years and the benchmark year (i.e., answers to the first two questions in the PyIO session of MPM).

5.4. Extraction Method

The extraction method in input-output system was initially suggested by Strassert (1968) and Schultz (1976, 1977). The method analyzes the importance of a sector or region by hypothetically extracting that particular sector or region from the input-output system – what would happen to the structure of the economy if the sector "disappeared." The output differences between the with and without that region or sector are then analyzed, and are generally considered as the importance of the extracted element. Several measures have been proposed in the literature to actually quantify the output differences (e.g., Cella 1984, Clements 1990, Dietzenbacher et al. 1993, and Dietzenbacher and van der Linden 1997).

The code presented here computes the backward linkage and forward linkage of the extraction method as outlined in Dietzenbacher et al. (1993). The importance of a sector or region is presented in terms of the backward and forward linkages between a system with and without the extracted element. Further, the backward linkage is computed in terms of the Leontief inverse while the forward linkage is obtained using the Ghoshian system.

The output difference between the full and the extracted system can be estimated from the following equation (Dietzenbacher et al. 1993):

$$\mathbf{x}-\overline{\mathbf{x}}=\begin{pmatrix}\mathbf{x}^{1}-\overline{\mathbf{x}}^{1}\\ \mathbf{x}^{R}-\overline{\mathbf{x}}^{R}\end{pmatrix}=\left\{\begin{bmatrix}\mathbf{L}^{11} & \mathbf{L}^{1R}\\ \mathbf{L}^{R1} & \mathbf{L}^{RR}\end{bmatrix}-\begin{bmatrix}(\mathbf{I}-\mathbf{A}^{11})^{-1} & 0\\ 0 & (\mathbf{I}-\mathbf{A}^{RR})^{-1}\end{bmatrix}\right\}\begin{pmatrix}\mathbf{f}^{1}\\ \mathbf{f}^{R}\end{pmatrix} \tag{16}$$

where $\mathbf{x}$ denotes output, $\mathbf{L}$ is the Leontief inverse matrix, $\mathbf{A}$ is the input requirement matrix, $\mathbf{f}$ is the final demand vector, superscript '1' and 'R' denotes the extracted region or sector and the rest of the system, respectively.

The above measure pertains to the backward linkage of the impact. In terms of the forward linkage, the difference is as follows

$$\left(\mathbf{x}-\overline{\mathbf{x}}\right)'=\begin{pmatrix}\mathbf{v}^{1\prime} & \mathbf{v}^{R\prime}\end{pmatrix}\left\{\begin{bmatrix}\mathbf{G}^{11} & \mathbf{G}^{1R}\\ \mathbf{G}^{R1} & \mathbf{G}^{RR}\end{bmatrix}-\begin{bmatrix}(\mathbf{I}-\mathbf{B}^{11})^{-1} & 0\\ 0 & (\mathbf{I}-\mathbf{B}^{RR})^{-1}\end{bmatrix}\right\} \tag{17}$$

where $\mathbf{v}$ denotes the primary input vector, $\mathbf{G}$ is the Ghoshian inverse, $\mathbf{B}$ is the output allocation matrix, and the rest is as previously defined. See Dietzenbacher et al. (1993) for further elaboration of the extraction method.

The extraction method in PyIO requires an input-output text file with complete information, i.e., containing identifiers, transactions, output, final demand and primary input. When the input-

output file contains a national (or one-region) input-output matrix, the function will do sectoral extraction: extracting one sector at a time to compute the extraction impact of that sector. If the input-output data is actually an interregional input-output table, the function will do regional extraction: extracting one region at a time to compute the extraction impact of that region. Note once again that the interregional input-output dataset should be inputted in such a way that the sectoral index moves faster than the regional index.

For this extraction method, results can only be seen in Excel. So, users need to say 'yes' for output in Excel. Two Excel worksheets will be loaded: one for backward linkage and the other for forward linkage computations. In the header of these two worksheet is written some background of the input-output, i.e., whether it is a national (or, one-region) or interregional input-output and the number of sector/regions in the data.

The structure of output should be read in the following way. Let us use the national input-output table as is inputted by `datafile.txt`. See the screen drop below:

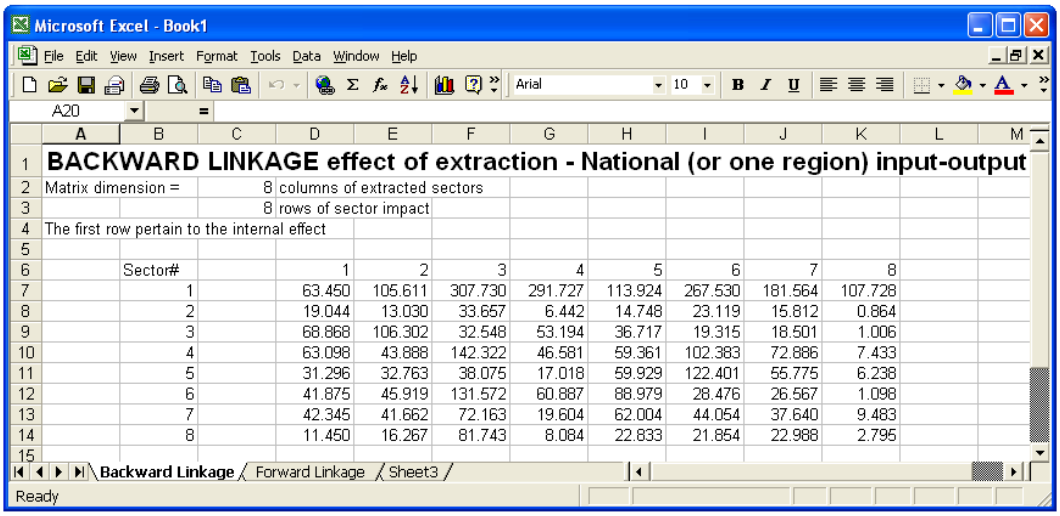


Figure 12. Screen drop from extraction method

Each of the columns on the above matrix provides the sectoral impact if the sector in that column is hypothetically extracted from the input-output system. However, the first row indicates the impact on its own sector. This result has a direct link to the equation (16), where the extracted sector is placed as the first element in the impact matrix $\mathbf{x} - \overline{\mathbf{x}}$. In this sense, the sector # given in the leftmost column should be read cautiously.

The first column gives the output impact when sector 1 is hypothetically extracted from the system. Thus, 63.450 is the impact to sector 1 when that sector is hypothetically extracted. Dietzenbacher et al (1993) called this the feedback effect. Also, 19.044 in the first column indicates the impact to sector 2 when sector 1 is extracted. Likewise, the second column gives the sectoral impact if sector 2 is extracted from the input-output system. The first element in this column is the impact to sector 2 when sector 2 itself is extracted from the system. The next element, 13.030 is the impact to sector 1 when sector 2 is extracted, and the next element 106.302 is the impact to sector 3 when sector 2 is extracted and so forth.

The result for the forward linkage computations should be read in a similar manner as the backward linkage computations.

The same cautions should also be used when dealing with this analysis when applied to an interregional input-output dataset. As mentioned earlier, in the interregional framework, the extraction works by each region, not by sector. Therefore, the output impact will comprise *r* columns, where *r* is the number of regions in the system; and d rows, where *d=r*n* is the number of input-output dimension. The first *n* rows of the matrix pertain to the own-region impact of extracting the region from the interregional system.

5.5. Pull-push analysis

Pull-push analysis is a flow decomposition method, based on the superposition principle, that examines the degree to which the structure of flows might be decomposed into a set of subflows (Sonis 1980; Jackson et al. 1989), each of which acts according to extreme tendencies. That is, a given flow matrix $\mathbf{Y}$ can be rewritten as a weighted sum of some extreme tendencies matrices:

$$\mathbf{Y} = \sum_{i=1}^{n} p_i \mathbf{X}_i = p_1 \mathbf{X}_1 + p_2 \mathbf{X}_2 + \cdots + p_n \mathbf{X}_n \qquad (18)$$

where $0 \le p_1 < p_2 < \cdots < p_n \le 1$ and $\sum_{i=1}^{n} p_i = 1$

In this fashion, each extreme tendency can be written as a binary matrix describing the inter-sectoral relationship in the hierarchy of decompositions. In the context of input-output, pull analysis can show the backward linkage patterns in an economy, while push analysis reveals the patterns of forward linkage patterns.

In PyIO, the push-pull analysis requires an input-output data that contains at the minimum the two identifiers and transaction matrix. As usual, users will be prompted whether output in Microsoft Excel is preferable. If so, a Microsoft Excel file will be automatically opened.

For the illustration below, we will use the `datafile.txt` in the push-pull analysis. For each analysis, they will be written in four worksheets. Figure 11 shows the example of these worksheets for pull analysis (the Microsoft Excel results for push analysis follows exactly the same structure.) The first worksheet in the Excel file is named "`Flow matrix`" showing different decomposed flow matrices according to the data set. The weight of each decomposed flow is written under each decomposed flow matrix together with its cumulative weights (CP). The maximum of each column in each decomposed matrix are presented in dash-bordered cells in the matrix, and the minimum of the maximum (minimax) of the whole matrix is marked as cells in red. The second worksheet is the "`X-tendencies matrix`" showing the corresponding extreme tendency matrix for the decomposed flow matrix. Note that different decomposed extreme tendency matrices are marked by different colors. The third worksheet denoted as "`Cumulative X`" is the cumulative extreme tendency matrix, which is also colored according to different decomposed levels. The fourth worksheet is "`weights (p)`", showing all the weights of the decomposed extreme tendency matrices.

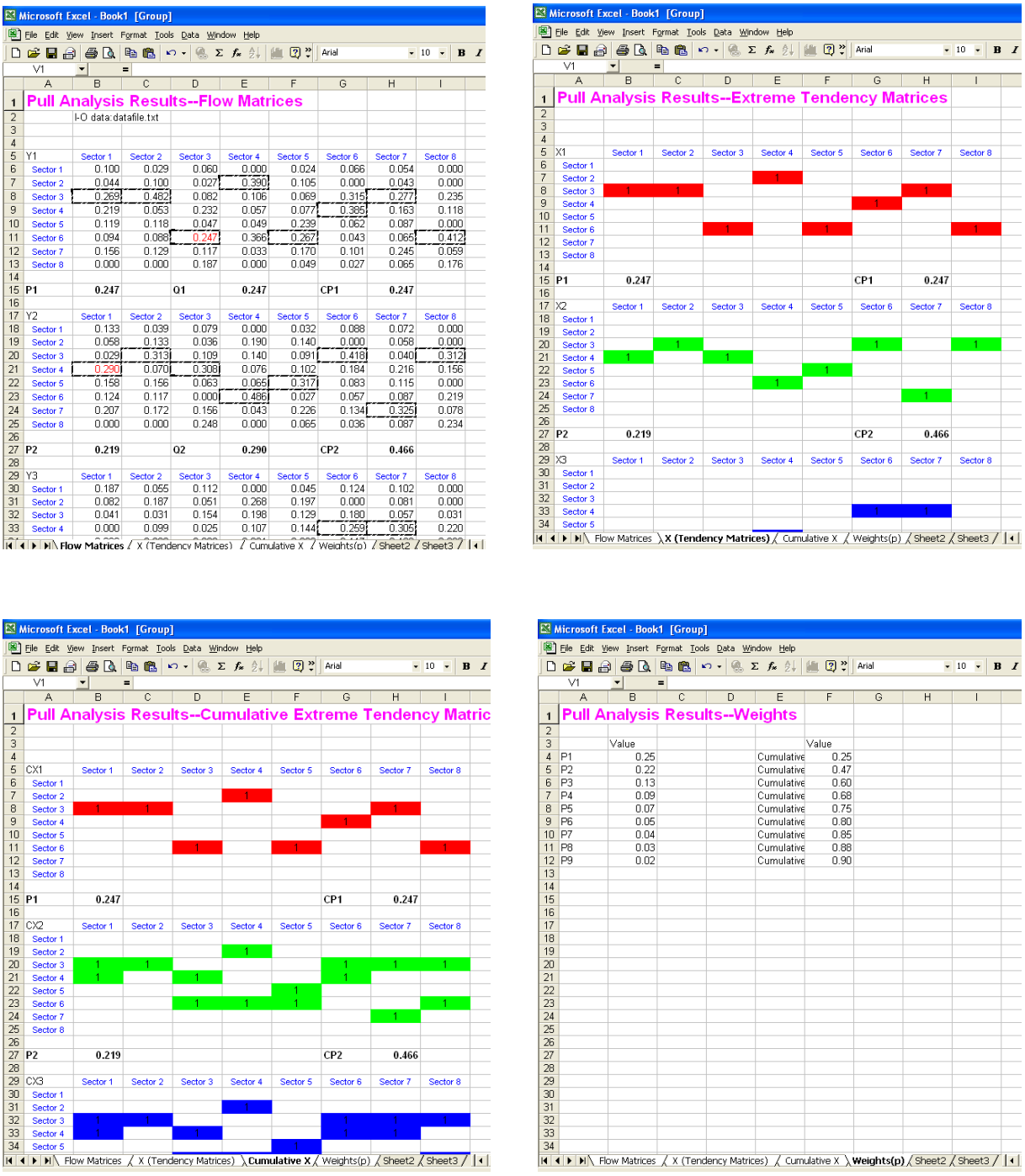


Figure 13. Screen drops from pull analysis

If users do not prefer the Microsoft Excel results, then all of the results will be written in an output text file. The structure of this output file will be similar to the above description.

5.6. Field of influence

The underlying idea of the field of influence is to assess the changes in the Leontief inverse matrix resulting from the change in one or more direct input coefficients in the inverse Leontieof matrix (i.e. total requirements matrix), research on the fields of influence applied to Input-Output models was initiated by M. Sonis and G.J.D. Hewings in the series of publications started from 1988. At this stage of development, PyIO computes only the first order field of influence.

The first-order field of influence deals with a single change in one element of the coefficient matrix **A**. For example, say that the changes take place at the element a_{ij}. The first order field of influence $F[(i,j)]$, of the incremental change e_{ij} is an nxn matrix generated by a multiplication of the j-th row of the matrix **B** with the i-th column (Okuyama et al. 2002):

$$F\left[(i,j)\right] = \begin{pmatrix} b_{1i} \\ \vdots \\ b_{ni} \end{pmatrix} \begin{pmatrix} b_{j1} & \dots & b_{jn} \end{pmatrix} \tag{19}$$

In the PyIO, after inputting the name of input-output text file, users will be asked to input the row and column associated with the source of change. Note that the number of row and column, for this purpose, starts from 1 (not 0). Users will also be prompted whether Excel output should be written. Otherwise an output text file will be created. If the primary input-output is named `datafile.txt` and the field of influence is computed for cell (3,8), then this output text file will be named `datafile_FOICell(3,8).out`.

Reference:

Cella G (1984). The Input-Output Measurement of Interindustry Linkages, *Oxford Bulletin of Economics and Statistics* 70: 705-12

Clements BJ (1990). On the Decomposition and Normalization of Interindustry Linkages, *Economics Letters* 33: 337-340

Dietzenbache E, van der Linden JA, Steenge AE (1993). The Regional Extraction Method: EC Input-Output Comparisons, *Economic Systems Research* 5: 185-206

Dietzenbacher E, van der Linden JA (1997). Sectoral and Spatial Linkages in the EC Production Structure, *Journal of Regional Science*: 235-258

Feldman SJ, McClain D, Palmer K (1987). Sources of structural change in the United States 1963-1978: an input-output perspective, *Review of Economics and Statistics* 69: 503-510

Jackson RW, Hewings GJD, Sonis M (1990). Decomposition Approaches to the Identification of Change in Regional Economies, *Economic Geography*: 216-231

Lutz M (2001). *Programming Python*. O'Reilly

Miller RE, Blair PD (1985). *Input-Output Analysis: Foundations and Extensions.* Prentice-Hall

Okuyama Y, Hewings GJD, Sonis M, Israilevich P (2002). Structural changes in the Chicago economy: A field of influence analysis. In: Hewings GJD, Sonis M (eds.) Trade, Network and Hierarchies: Modeling Regional and Interregional Economies, Springer Verlag, pp, 201-224

Rasmussen P (1956). *Studies in Inter_Sectoral Relations.* Copenhagen, Einar Harks

Sonis M (1980). Locational Push-Pull Analysis of Migration Streams, *Geographical Analysis* 12: 80-97

Sonis M, Hewings GJD, Guo J (1997). Comparative analysis of China's metropolitan economies: an input-output perspective. In: Chatterji M, Kaizhong Y (eds.) *Regional Science in Developing Countries*. Basingstoke, Macmillan Press:147-162

___, (1996). Sources of Structural Change in Input-output Systems: A Field of Influence Approach, *Economic Systems Research* 8:15-32

Sonis M, Hewings GJD (2004). *Evolving Spatial Economic Structures: Input-Output Analysis Perspectives* (web-based manuscript in preparation)

Strassert G (1968). Zur Bestimmung Stretegischer Sektoren mit Hilfe von Input-Output-Modellen, *Jahrbucher fur Nationalokonomie und Statistik* 182: 211- 215

Appendix: PyIO using functions

PyIO can also be used in the programming environment provided by Python. There are several of these environments. Two easily accessed environments are the PythonWin and IDLE. Users interested in using these environments can find further information about these in Python documentation that are provided upon its installation. Here we will assume that users have had sufficient background on these environments.

This environment is suitable for more advanced programmers who want to explore PyIO at a technical level. One of the advantages of using the programming environment is the possibility to (needless to say) program. One useful application is, for example, if one has series of tasks on the input-output analysis. Rather than executing the analysis one by one through the `run_pyio.py`, one can instead write another Python code that automatically execute the whole tasks at once. This makes the whole analysis more efficient.

PyIO comprises a set of functions, written in the `pyio.py` file. The IDLE environment is easily accessed by right clicking this file, and then click "`Edit with IDLE`". Of course, Python must be properly installed for that option to appear by right clicking. Two windows will appear, and the active one will show the Python code. Users can run this code by clicking "`Menu – Run Script`" or simply by pressing `Ctrl-F5` simultaneously. Having done that, the second window will become active, and users are ready to use PyIO functions at the prompt. We will discuss these functions below, after showing how the other environment, i.e., PythonWin, can be called.

Once the Python is successfully installed, one can call PythonWin icon in the Windows Start menu. Once it is loaded (look at the screen drop below in figure A.1), there are several ways to load the PyIO module: (1) Run the module by clicking File-Run command menu as shown; (2) Run the module by pressing `Ctrl-R`; or (3) Click the run icon (a person on the run) on the menu bar. In all of these three ways, you will need to browse to the directory where the PyIO is located, the same directory where you should put all of your input-output data files for analysis. Detailed descriptions of the functions contained in PyIO are given in the subsequent section.

A typical example of a PyIO session using functions can be seen below. Let's replicate the example shown in section 2.4, calculating the output impact analysis. The same analysis can be done by inputting the following command in the PythonWin (or IDLE) prompt:

```
>>>  ImpactAnalysis('datafile.txt','sce.txt')
```

Having entered the command, users will be prompted if they want the results printed in Excel. When all result writing is done, users can see that an array is returned in PythonWin window. The array is the result of computation. Given there are eight sectors and three final demand scenarios in `sce.txt`, the `ImpactAnalysis` function returns an array of dimension 3x8. Look at figure A.2. below.

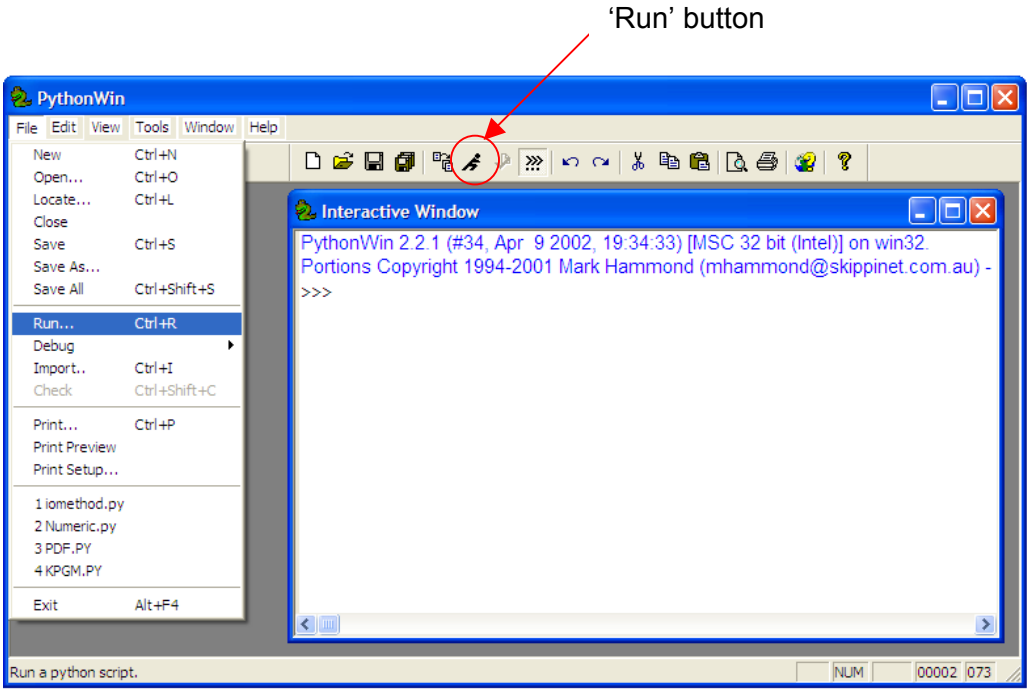


Figure A.1. Opening Pythonwin and uploading (or, running) a module

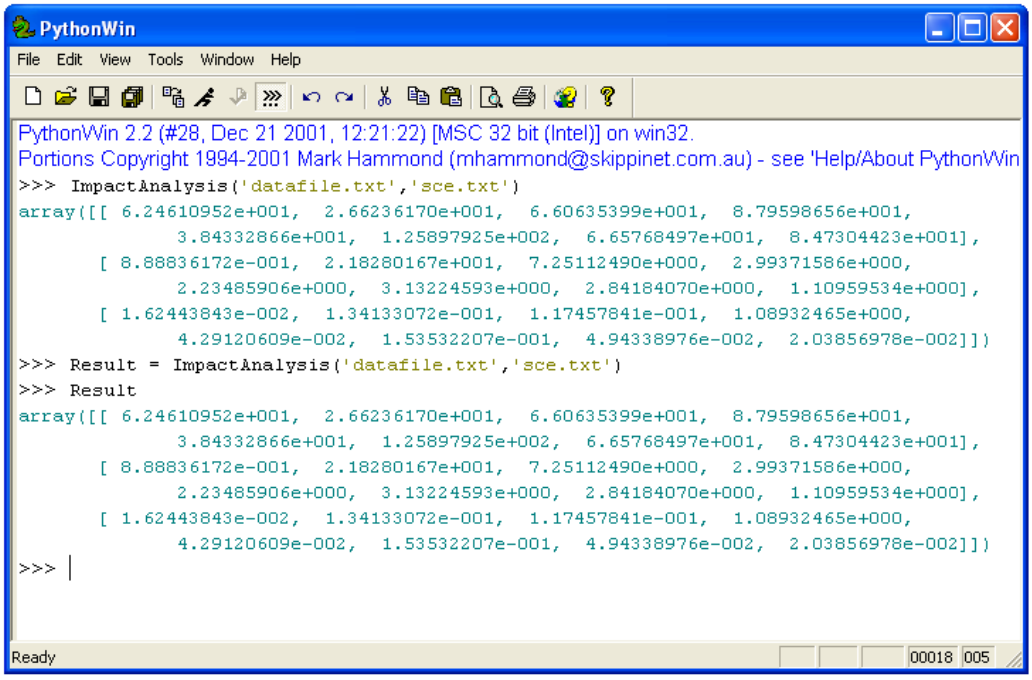


Figure A.2. Typical session using PyIO functions

Also shown in figure A.2. is another way to carry out a PyIO session with function. Users can actually assign a name to the output of a function. In the example above, the output of `ImpactAnalysis` is stored in a variable named `Result`. This way, Python does not automatically show the resulting array upon computation. Users can call the output by writing down the name of variable. This `Result` is ready for further manipulation in Python.

The complete list of PyIO functions can be seen in table A.2. Typically, these functions require the primary input-output data text file, and in some cases, together with the additional information. Refer back to the earlier discussion about each of the analysis, for the appropriate structure of the additional information text file.

In the above example, `ImpactAnalysis` only returns one output: an array of 3x8 dimension. Some other functions return more than one output. To call such output, it is important to know about indexing in Python, i.e., about how elements are placed in an array. Many times in Python, we are dealing with lists or arrays, whose elements can be numbers or strings (letter or words). The location of each element is marked by an index, which in Python starts with '`0`'. See an illustration below:

```
>>> mylist = [9,6,2,5,7]
>>> mylist[0]
9
>>> mylist[3]
5
>>> mylist[-1]
7
```

The first syntax gives a very simple list, which is named `mylist`. The first element of `mylist` is called by inputting the index '0.' Note also that Python calls the last element of the list with an index '`-1`.' For further elaboration of indexing, readers may want to consult a more formal Python documentation. Readers may want to get familiar with this indexing because in most cases the functions in this module will return an array. Knowledge of this indexing will be important to retrieve desired result from any array.

Therefore, coming back to our `Result` from figure A.2, users can write:

```
>>> Result[1]
```

to get the vector of output impacts of the second scenario.

Some other functions in PyIO return a multiple array. For example, the `AggregatingSector` function returns two items: new (aggregated) transaction matrix and new (aggregated) output vector. Thus, if users want to carry out some aggregation and want to see just the new transaction matrix, this can be done by the following:

```
>>>  result = AggregatingSector('io.txt','aggregating_info.txt')
>>>  newmatrix = result[0]
```

The complete list of functions and items that are returned from PyIO functions can be seen in table A.2.

Table A.1. Glossary of analysis, functions, and files in PyIO*

Name of analysis**	Additional information (in a text file)	Name of output file created***	Name of new input-output data file
Table operations			
1-1 Sectoral aggregations	Sectoral aggregation info (e.g., `sec_agr.txt`)	`io_Aggregation.out`	`io_new.txt`
1-2 Spatial aggregations	Regional aggregation info (e.g., `reg_agr.txt`)	`io_Aggregation.out`	`io_new.txt`
1-3 Updating table by RAS	Sum of intermediate output (U), sum of intermediate input (V), and total output (X) (e.g., `info.txt`)	`io_RASUpdate.out`	`info_RAS.txt`
1-4 Updating table by RSP	Regional export, import, output (e.g., `reg_a.txt`)	`io_RSPUpdate.out`	`reg_a_RSP.txt`
1-5 Updating table by SLQ	National, regional variable (e.g., `reg_b.txt`)	`io_SLQUpdate.out`	`reg_b_SLQ.txt`
Basic I-O Analysis			
2-1 Leontief inverse	(none)	`io_LeontInv.out`	(none)
2-2 Ghoshian Inverse	(none)	`io_GhoshInv.out`	(none)
2-3 Impact Analysis	Final demand scenario(s) (e.g., `sce.txt`)	`sce_Impact.out`	(none)
2-4 Output multiplier	(none)	`io_OutMult.out`	(none)
2-5 Income multiplier	Sectoral income (e.g., `income.txt`)	`io_IncMult.out`	(none)
2-6 Employment multiplier	Sectoral employment (e.g., `employ.txt`)	`io_EmpMult.out`	(none)
2-7 Input (or supply) multiplier	(none)	`io_InpMult.out`	(none)
Advanced I-O Analysis			
3-1 Key sector analysis	(none)	`io_KeySector.out`	(none)
3-2 Output change decomposition	I-O data of different times (e.g., `io_t.txt` & `io_0.txt`)	`io_t_Decomp.out`	(none)
3-3 Multiplier product matrix	(none)	`io_MPM(`no. of years, benchmark year`).out`	(none)
3-4 Extraction Method	(none)	none	(none)
3-5 Pull analysis	(none)	`io_Pull.out`	(none)
3-6 Push analysis	(none)	`io_Push.out`	(none)
3-7 Field of Influence	Cell position (e.g., `(r,s)`)	`io_FOICell(r,s).out`	(none)

Note: * Suppose that the input-output datafile is named `io.txt`; **Numbers preceding the name of analysis refer to those that users need to input to get to that analysis; *** These files are created when users answer 'no' for results in Excel.

Table A.2. Glossary of analysis, functions, and returns in Python*

Name of analysis**	Syntax to use in PythonWin or IDLE***	Returns
Table operations		
1-1 Sectoral aggregations	`AggregatingSector('io.txt','sec_agr.txt')`	New transaction matrix and new output vector
1-2 Spatial aggregations	`AggregatingRegion('io.txt','reg_agr.txt')`	New transaction matrix and new output vector
1-3 Updating table by RAS	`UpdateRAS('io.txt','info.txt')`	New A matrix
1-4 Updating table by RSP	`UpdateRSP('io.txt','reg_a.txt')`	New A matrix
1-5 Updating table by SLQ	`UpdateSLQ('io.txt','reg_b.txt')`	New A matrix
Basic I-O Analysis		
2-1 Leontief inverse	`LeontiefInverse('io.txt')`	Leontief inverse matrix
2-2 Ghoshian Inverse	`GhoshianInverse('io.txt')`	Ghoshian inverse matrix
2-3 Impact Analysis	`ImpactAnalysis('io.txt','sce.txt')`	Matrix of output impact
2-4 Output multiplier	`OutputMultiplier('io.txt')`	Output multiplier
2-5 Income multiplier	`IncomeMultiplier('io.txt','income.txt')`	Income multiplier
2-6 Employment multiplier	`EmploymentMultiplier('io.txt','employ.txt')`	Employment multiplier
2-7 Input (or supply) multiplier	`InputMultiplier('io.txt')`	Input multiplier
Advanced I-O Analysis		
3-1 Key sector analysis	`KeySector('io.txt')`	A list containing two sorted sector vectors according to descending forward linkage and backward linkage respectively. Another list containing two vectors of descending forward linkage and backward linkage respectively.
3-2 Output change decomposition	`Decomposition('io_t.txt','io_0.txt')`	A matrix containing the decomposition parts in the following order: decom_1, decom_2, decom_3, selfdecom_1, selfdecom_2, selfdecom_3, nonselfdecom_1, nonselfdecom_2, nonselfdecom_3, total
3-3 Multiplier product matrix	`MPMAnalysis(`no. of years`,` benchmark year`)`	List of sorted MPM (in their descending sector order), descending row order, descending column order, number of sectors, unsorted MPM. This list is stacked from the initial to the last years.
3-4 Extraction Method	`ExtractionMethod('io.txt')`	Two matrices of extraction impact: backward linkage, forward linkage
3-5 Pull analysis	`PullAnalysis('io.txt')`	p-value vector
3-6 Push analysis	`PushAnalysis('io.txt')`	p-value vector
3-7 Field of Influence	`FOI1('io.txt', (r,s))`	Field of influence matrix

Note: * Suppose that the input-output datafile is named `io.txt`; ** Numbers preceding the name of analysis refer to those that users need to input to get to that analysis; *** Quotation marks to denote file names are required. See discussion in Appendix for more information about functions in PyIO.