

Introducing IPAT-S

Scenario Scripting Language

Eric Kemp-Benedict

March 2006

© 2005, 2006 Eric Kemp-Benedict

Introducing IPAT-S Scenario Scripting Language may be freely printed, copied, and distributed in physical or electronic form.

The layout for this document is based on a design by Mirto Silvio Busico (m.busico@ieee.org) as an OpenOffice.org Writer template.

Index

1	What is IPAT-S?.....	2
1.1	IPAT Studio.....	2
2	IPAT-S Basics.....	4
2.1	Years.....	4
2.2	Dimensions.....	4
2.3	Variables.....	5
	Summable variables and variables.....	5
	Ratios.....	6
	Logicals and Numbers.....	7
2.4	Chain Expressions.....	7
	Why are they called “chain expressions”?.....	8
	A note on dimensions.....	8
2.5	Linear programs.....	9
2.6	Output.....	9
	Report Statements.....	9
	Print Blocks.....	10
2.7	Making scripts more clear.....	11
	Using “ditto”.....	11
	Comments.....	11
	Key Inputs.....	12
3	A Sample IPAT-S Script.....	13
3.1	The script.....	13
	Lines 1-13: The comment block.....	14
	Lines 15-16: Year declaration.....	15
	Lines 18-19: Dimension declarations.....	15
	Lines 21-29: Variable declarations.....	15
	Lines 31-47 and 57-70: Initializing variables.....	16
	Lines 49-55 and 72-83: Calculating and reporting indicators.....	17
3.2	Discussion.....	18
4	Back of the Envelope Plus.....	20
4.1	Exhausting Fuel Resources.....	20
	The basic script.....	20
	Enhancing the script.....	22
4.2	Discussion.....	26
5	Recipes.....	27
5.1	IPAT and ImPACT.....	27
5.2	Explicit Time Dependence.....	31
	Time-dependent chain expressions.....	31
	Logistic curves and “loglets”.....	32
5.3	Intensities.....	33
	Benchmarks.....	33
	Income and price elasticities.....	34
	Kuznets Curves.....	35
	Asymptotic curves.....	37

1 What is IPAT-S?

What IPAT-S is can be summed up in a sentence: it is a scripting language for quantitative scenario analysis designed to support rapid, participatory scenario development for sustainability studies. It could also be called a *domain language* – that is, a programming language designed to support tasks in a particular domain, in this case scenario development. Finally, it could be called a *modeling language*, a programming language for building mathematical models.

A *scripting language* is a high-level, interpreted programming language. Programs written in a scripting language are generally called “scripts.” Scripting languages are often intended for a special purpose and are usually intended for rapid development. An *interpreted language* is one that requires another program – the interpreter – to run a script. In the case of IPAT-S, all of the software and documentation is available for free from the IPAT-S web site, <http://ipat-s.kb-creative.net/>, so anyone with an Internet connection can download the interpreter and associated software.

A *scenario* (as used in this book) is a story of the future used to inform current decision-making. Scenario analysis can be used for policy development as well as education and outreach. Many scenario exercises include a quantitative component; this is the component that IPAT-S is designed for. For more information about scenarios in general, see the web site <http://www.scenariosforsustainability.org/>.

Sustainability is a nebulous but often-used term to describe patterns of consumption and production in some area – a city, a watershed, a region, the world – that are capable of being maintained indefinitely without dangerously degrading the ecological and social processes on which the system relies. The concept usually includes a notion of social justice, on the grounds that an unjust system is not capable of being maintained indefinitely.

IPAT-S is designed for simplicity, legibility, flexibility, speed and openness. IPAT-S is an open-source project, hosted on SourceForge (<http://sourceforge.net>). The interpreter is written in C, for speed of execution, and has a compact but clear syntax, for speed of model development.

1.1 IPAT Studio

IPAT-S by itself is just a language – a set of rules for expressing a scenario developer’s intentions. For a script to actually do something, it must be run through the interpreter, which is a small, stand-alone command line program. However, when developing scripts, working at the command line can be a challenge. For easy script development, there is a free, open-source integrated development environment (IDE) for IPAT-S called IPAT Studio. In this book it is assumed that you will be using IPAT Studio to run the scripts.

IPAT Studio is part of the normal IPAT-S installation. After installing the program, start IPAT Studio, and you should see a window like that in Illustration 1. As shown in the illustration, when the program is first run, it opens a sample script titled `Default.ips`, which demonstrates many of the features of the IPAT-S language.

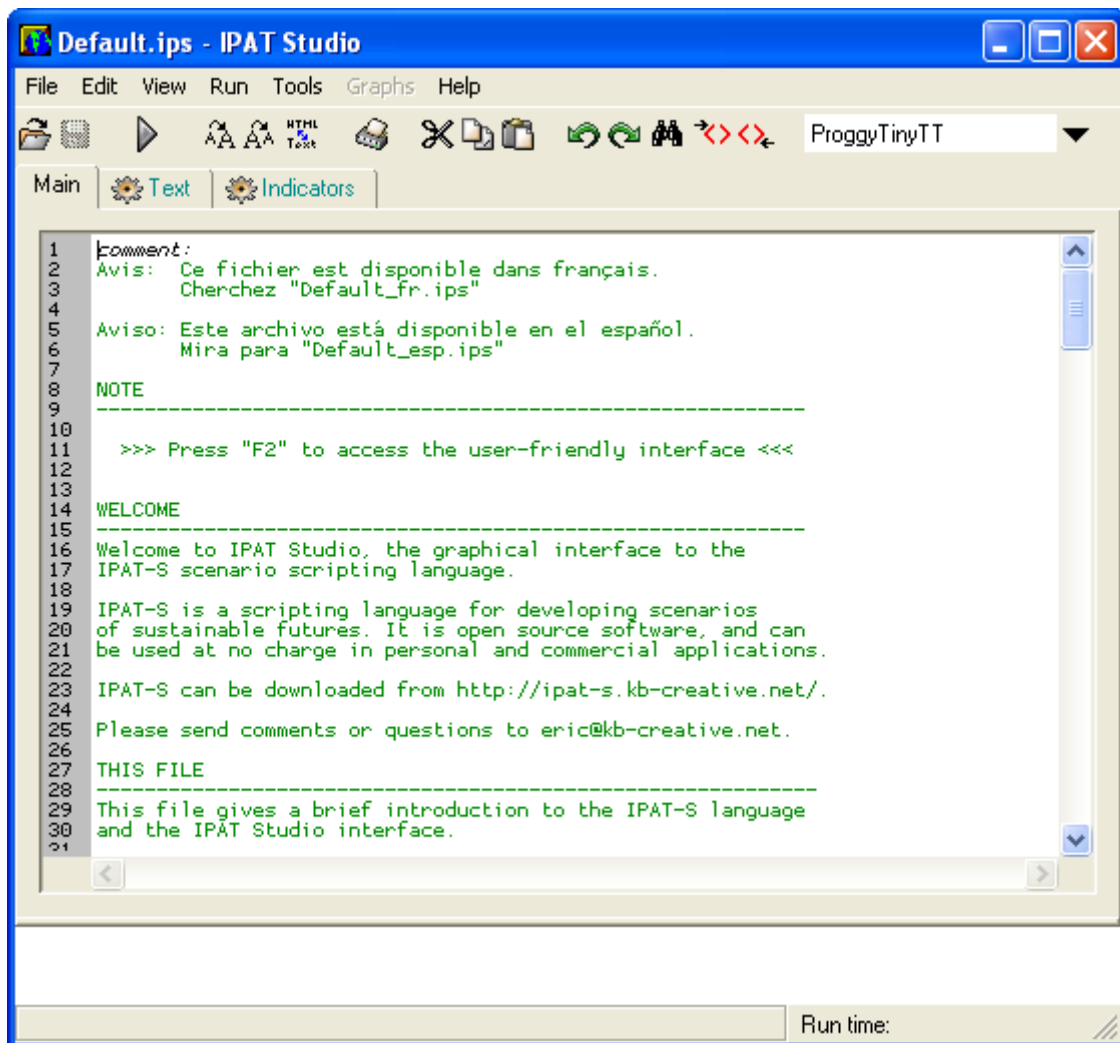


Illustration 1: IPAT Studio

To run the default script in IPAT Studio, click on either the Text or the Indicators tab. The script will run and the appropriate output will be displayed.

For more information about IPAT Studio, browse the help file that comes with the program. You can get a quick introduction to IPAT-S by viewing and running the sample scripts that are included in the IPAT-S distribution.

2 IPAT-S Basics

IPAT-S is a scripting language designed to support scenario development. Many of the language elements reflect its basic purpose, while additional elements reflect other goals, such as speed and flexibility, or the readability of the script.

For the most part, IPAT-S is a traditional imperative programming language, meaning that the script writer instructs the computer how to calculate the model. IPAT-S also has one *declarative* programming construct, the LP block. In an LP block, a linear programming problem is specified and the IPAT-S interpreter figures out how to solve it. However, the block as a whole is calculated as soon as it is encountered in the script, so it fits inside the overall imperative structure.

There are several sources of information about the IPAT-S language. It is documented in the *IPAT-S Language Reference*, a good source for in-depth information that is available as a PDF document in the standard IPAT-S distribution. Also, a quick reference to the language is available in the IPAT Studio help file. Finally, numerous sample scripts are distributed with the program. With so many sources of information available, the aims of this chapter will be modest – to provide a brief survey to the most important aspects of the language in order to launch into writing scripts with a solid base.

2.1 Years

Most scenarios start from a reference *base year*, a year for which most of the required data are available. For practical reasons, the base year is usually two to five years before the year of the study, because data take time to prepare and release. Scenarios are usually presented for one or more *scenario years*. They might be developed for every year between the base year and the final scenario year, or may be developed only for a handful of years.

In IPAT-S, all scripts must start with a declaration of years. The scenario years are optional, while the base year is required, but in nearly every case a script will have both base years and scenario years.

Here is an example of a year declaration:

```
base year 2000
scenario years 2010 2020 2030 2040 2050
```

This could also have been written¹

```
base year 2000
scenario years 2010 to 2050 by 10
```

2.2 Dimensions

The numbers in a scenario study can usually be given meaningful labels. For example, *energy use* might be labeled by end-use sector – households, industry, agriculture, transport

¹ There are often shorthand ways to write things. In the design of the IPAT-S language, a premium was placed on rapid development of scripts, and shorthand alternatives are one way to accomplish this.

and services. One of the most common labels is *scenario* itself. In most scenario studies, several scenarios are prepared, and the entire data structure is generally (although not always) the same in each one.²

In IPAT-S, labels for variables are called *dimensions*. Dimensions are optional, but if they are present, they must be declared immediately after the year declarations. Dimension declarations start with the keyword `dimension`, followed by a dimension *name* made up of letters, digits and underscores (“_”), followed by a list of dimension *labels*. Dimension names cannot start with a digit, and cannot contain any spaces, while dimension labels can contain any combination of characters. Some examples:

```
dimension scenario 'Business as Usual' 'High Growth' 'Stagnation'
```

It is not necessary to spell the word “dimension” out:

```
dim EnergySector 'Households', 'Industry', 'Agriculture', \
    'Transport', 'Services'
```

In this example, the line-continuation character, a backslash, was used to extend the long line onto a second line. The example also demonstrates the use of optional commas between dimension labels.

The dimension labels are arbitrary strings, and can contain any characters. They can be delimited by either single or double quotes, as in this example:

```
dim country "Mali" "Burkina Faso" "Cote d'Ivoire"
```

2.3 Variables

IPAT-S offers five data types: *summable variables*, *variables*, *ratios*, *logicals* and *numbers*. IPAT-S also has strings, but they only appear as string constants – there are no (user-defined) string variables.

Summable variables and variables

The workhorses of IPAT-S are the summable variable and variable data types. These contain double-precision floating-point numbers labeled by year and dimension. From the point of view of the interpreter, there is no distinction between summable variables and variables. The distinction is for anyone who reads the script.³

A summable variable is a number that it makes sense to add up and that should be labeled by year and dimension, such as population, GDP, or total energy use.

A variable is any other number that should be labeled by year and dimension, such as a fuel share.

² In programming parlance, IPAT-S variables are represented by multidimensional *associative arrays* or *hash maps*.

³ IPAT-S offers the script writer several self-documenting options, in addition to comments. These options are for human readers, not for the interpreter – they are intended to support external model reviews, long-term maintenance of scripts, and sharing of scripts.

When variables are declared, any dimensions associated with them must be specified. Variable names can consist of letters, digits and underscores, but cannot start with a digit. Some examples:

```
sumnable variable GDP{scenario, region} Population{region}

summvar Fuel_Use{scenario, region, sector}, \
        Transport_Volume{scenario, region}

variable FuelShare{sector}

var Poverty_Line{scenario}
```

As seen in these examples, it is not necessary to fully spell out “summable variable” or “variable.” Also, the use of commas to separate variables is optional.

Variables and summable variables are initialized using a more or less conventional syntax. For example:

```
GDP.0{region='North Region'} = 1520
```

In this expression, the `.0` refers to the base year, and the `region` dimension has been specified. It is possible to initialize values for several years at once and for several dimensions at once: see the *Language Reference* for details.

In contrast to initialization, which uses a standard equals sign (“=”) to assign the results of calculations to a variable, scenario calculations involving summable variables or variables use a special syntax called *chain notation*. This special syntax is described later in this chapter.

Ratios

In quantitative scenario analysis, one of the most common calculations to perform is to apply a growth rate to a base-year value. For example, GDP might be calculated by specifying base-year GDP and a growth rate. Unlike summable variables and variables, which are specified for each year, a growth rate *connects* years. In IPAT-S this connecting role is carried by *ratio* variables.

As with variables and summable variables, ratio variables can be labeled with dimensions:

```
ratio income{scenario, region}
```

However, ratio variables are initialized and combined using a different syntax than summable variables or variables. For example:

```
income{scenario='BaU', region='North Region'} = growth(2.3%)
```

specifies the value of `income` as an *index* that starts at a value of 1.0 in the base year and grows at a rate of 2.3% per year throughout the scenario. There are variations on this syntax that are described in the *Language Reference* and in the IPAT Studio help file.

Why use ratio variables? Although they are less flexible than variables or summable variables, they can be very useful in creating compact, readable, maintainable scripts. In general, if the easiest way to specify a change in a scenario variable is through a growth rate or relative change, it should be represented by a ratio variable.

Logicals and Numbers

Logical variables and number variables each contain a single value – they are not labeled by year or dimension. An example of the use of a number variable would be a conversion factor that is used throughout a script, while a logical value might contain a flag that turns a block of code on or off. See the *Language Reference* or the IPAT Studio help file for the syntax, and the sample scripts for examples.

2.4 Chain Expressions

If the workhorse variables in IPAT-S are the summable variables and variables, then the workhorse form of calculation is the *chain expression*. Chain expressions are used to calculate the values of summable variables and variables in IPAT-S scripts.

There are two forms of the chain expression. The first has the general form⁴

```
:: expression >> ratio expression -> variable
```

Before entering a chain expression of this form in a script, the *variable* should be initialized to have some base year value. The chain expression does not change this base-year value. Instead, it constructs an index out of the *expression* and the *ratio expression* and applies that index to the *variable*.

An example can help make this clear. Suppose that population is specified explicitly for all years, while GDP is specified only in the base year and is otherwise calculated by supplying a GDP/capita growth rate of, say, 1.2% per year. A sample script that would cover this case is:

```
base year 2000
scenario years 2010 2020 2030
summvar Population = 125, 135, 142, 144
GDP.0 = 1270
:: Population >>gr(1.2%) -> GDP
```

In this script, GDP is calculated starting from its base-year value and then having it grow as a result of both population growth and GDP/capita growth.

Although this syntax may seem obscure, it is very compact and with a little practice can make scripts much easier to write and maintain than if they were written in a language that does not provide ratio variables or chain expressions. In such a language, the calculation above would probably involve the following steps (after first specifying the scenario values for population and the base-year value for GDP):

1. Calculate GDP/Capita in the base year.

⁴ There are many variations on this basic form – see the *Language Reference* or the on-line help for IPAT Studio.

2. Calculate GDP/Capita in the scenario by applying a growth rate.
3. Multiply population in each year by GDP/Capita.

The chain notation takes care of all of these details, which in a typical scenario calculation would be repeated many times.



While calculating a variable or summable variable by applying an index or growth rate to it is very common in quantitative scenario analysis, it does not cover all cases. Sometimes what is needed is just an ordinary assignment statement, like the equals sign in conventional programming languages. The second form of the chain expression covers this case. It has the general form

```
:: expression -> variable
```

With this form of chain expression, the *variable* is assigned the result of calculating the *expression* in all scenario years.⁵

Why are they called “chain expressions”?

Chain expressions can be linked together one after another to compact a sequence of calculations into one expression. That is why they are called “chain expressions” – they can be combined like links in a chain.

A typical example would be an expression that calculates both GDP and an environmental impact:

```
:: Pop >> income -> GDP >> intensity -> Impact
```

In this expression, *income* and *intensity* are ratio variables, while *Pop*, *GDP* and *Impact* are summable variables.

A note on dimensions

In the expressions above, the variables might or might not have dimensions. In most cases it is not necessary to specify dimensions in chain expressions – the IPAT-S interpreter handles them automatically. The interpreter also handles situations in which the dimensions of the variable do not match the dimensions in the expression being calculated – see the *Language Reference* or the on-line help for more information.

By automatically handling dimensions, the IPAT-S language makes it relatively simple to extend a script and reuse pieces of scripts. Since the dimensions do not have to be specified explicitly in chain expressions, the dimensions and dimension values can in many cases be changed without changing the expressions that calculate scenario values.

⁵ Why not use an ordinary equals sign? For two reasons: because the equals sign is reserved for initializing summable variables and variables (e.g., to their base-year values), and to maintain the convention that chain expressions are used to assign scenario values to variables and summable variables.

2.5 Linear programs

Achieving sustainable development is a challenge in large part because of the presence of constraints – limits on resources, on the waste-absorbing capacity of the environment, on the rate of change of key social systems. A sustainability analysis typically explores the range of options consistent with a set of constraints.

For constraints that can be expressed mathematically, a key tool for quantitative analysts faced with a constrained system is *mathematical programming*. Mathematical programming problems are characterized by a set of constraint equations and an *objective function* that expresses the objective to be met: the mathematical programming algorithm seeks to maximize or minimize the objective function subject to the constraints. For sustainability problems, a subset of mathematical programming, *goal programming*, is especially appropriate. In a goal program, the problem is specified by defining the constraints and a set of *goals* – the objective function measures how far away from the goal the system is (as a weighted sum of deviations from the goal), and the objective is to minimize that distance.

The full range of mathematical programming tools includes, in order of increasing complexity, *linear programming*, *quadratic programming* and *nonlinear programming*. IPAT-S supports linear programming (LP), and can be used to solve linear goal programs.

There is an extensive discussion of linear programming problems in the *IPAT-S Language Reference*, and several example scripts demonstrate the syntax. For the purposes of this chapter, a simple example of a linear programming block (an LP block) should suffice:

```
LP:
  solve for Inputs, Outputs
  max price * Outputs - cost * Inputs
  cost * Inputs <= currentBudget
  Outputs - TransfCoefficients * Inputs = 0{output}
:LP
```

In this LP, revenue (income from selling outputs less expenditure on inputs) is maximized, subject to the constraints that current expenditure fit within the current budget and that total outputs is the set of outputs that can be produced by the inputs used. The notation `0 {output}` says that the equation applies separately to all values of the dimension `output` – that is, that for every output, the total amount produced is equal to what can be produced from the given inputs.

2.6 Output

Once the figures for a scenario are calculated, they can be output in a text format. IPAT-S offers two ways to produce formatted output – the *report statement* and the *print block*.

Report Statements

Using report statements is the easiest way to quickly produce output from scripts. They offer a simple syntax for building indicators out of script variables.

Report statements have the general syntax

```
report expression as 'label'
```

where *expression* is an expression involving summable variables, ratio variables and variables, and *label* is a string that labels the expression. As with strings anywhere in IPAT-S, either single or double quotes can be used for the label. For example,

```
report GDP/Pop as "Income ($/capita)"
```

The syntax for expressions in report statements is almost identical to the syntax for chain expressions, but report statements have some additional options. See the *IPAT-S Language Reference* or the IPAT Studio help file for more information.

Print Blocks

The report statement is good for quickly generating indicators, and for many projects it is the only kind of output that is needed. The output from report statements can be browsed in a spreadsheet-like table in IPAT Studio and graphs made of the indicators. The graphs can be saved and used in documents, and figures from the table can be copied and pasted into a full-feature spreadsheet like OpenOffice.org Calc or Microsoft Excel. However, in some cases more control over the output is needed. The print block can be used in these cases.

The IPAT-S print block allows for free-form formatted text with embedded calculations. For example,

```
print:
The value for GDP in the base year is $[GDP.0].
:print
```

Any text between `print:` and `:print` will be output verbatim as text, except for text between square brackets, `[]`, which contain calculations and other instructions to the interpreter. To output square brackets themselves (for example, for a bibliographic citation), use double brackets. For example,

```
print:
As reported in Rogers et al. [[2]], the value for GDP
in the base year, [y.0], is $[GDP.0].
:print
```

The output from this print block, assuming the base year is 2000 and the initial GDP is \$2,300, is

```
As reported in Rogers et al. [2], the value for GDP
in the base year, 2000, is $2300.
```

The print block is potentially quite powerful, because many programs, such as GISs and the IISD's Dashboard of Sustainability, use plain text input files. Some standard plain-text file formats include CSV (comma-separated variable), XML (extensible

markup language), HTML (hypertext markup language, the language of the World Wide Web) and RTF (rich text format, used by many word processors).

If you have installed the Dashboard of Sustainability and use the print block to create a Dashboard input file, then it can be launched in the Dashboard from inside IPAT Studio. See the IPAT Studio help file for more information.

2.7 Making scripts more clear

In any programming language, a programmer must make an effort to make his or her programs easy to read and maintain. Different languages offer more or less support for doing this. In addition to some shorthands for frequently-used language elements, IPAT-S offers features for maintaining the legibility of scripts.

Using “ditto”

Many programming languages allow the user to define *macros* – shorthands defined by the programmer that can be used to make programs less cluttered and easier to maintain. IPAT-S offers a kind of “instant macro” feature invoked using the keyword `ditto`. Here is an example:

```
ditto Region = 'North Region':
GDP.0{' '} = 2300
Pop.0{' '} = 53010
CO2.0{' '} = 11.3
```

In this example, the `ditto` keyword tells the interpreter to store the text “Region = 'North Region'” and to insert it anywhere `ditto` marks (' ') are encountered in the script.

Using `ditto` has at least two advantages. First, it makes scripts shorter and easier to read. Second, it makes scripts easier to maintain, because text can be changed in one place rather than in several places – in the example above, if “North Region” is renamed to “Northern Regions”, the change would have to be made in only one place.

Comments

There are two forms for comments in IPAT-S, *inline* and *block*. Inline comments start with a pound sign, “#”. For example,

```
# This is an in-line comment
GDP.0 = 2300 # This is another in-line comment
```

Block comments, which can stretch over many lines, are marked off by `comment :` and `: comment`. For example,

```
:comment
This is a block comment, stretching over
several lines.

This block comment can be used to introduce
a script, document a list of variables, or
```

```
contain any other multi-line comments.  
:comment
```

Key Inputs

The numbers in a quantitative scenario exercise generally fall into one of four categories – inputs, indicators, data and internal variables. Scripts are easier to understand if these different kinds of numbers are kept separate from each other.

In order to distinguish key *inputs* from the other kinds of numbers, IPAT-S allows text to be set off by angle brackets, "<>", for example,

```
GDP.0 = <2300>
```

The angle brackets are ignored by the IPAT-S interpreter itself, but are used by the IPAT Studio IDE.

3 A Sample IPAT-S Script

There are many features of IPAT-S beyond the ones listed in the previous chapter. However, interesting scripts can be built even with this brief list. This chapter will start by presenting a complete script that uses only the features already introduced. The rest of the chapter will be devoted to exploring the script and seeing what it does.

3.1 The script

The script that will be discussed in this chapter is one of the sample scripts distributed with the IPAT-S software. In the IPAT-S distribution, it is called `EnergyLadderLP.ips`. Interested readers might want to open the script in IPAT Studio in order to run it, browse it and edit it.

The script is also reproduced here, in its entirety. Line numbers have been added for convenience.

EnergyLadderLP.ips

```

1) comment:
2) This script implements a model inspired by the "energy ladder"
3) hypothesis. The hypothesis is that households at different income
4) levels climb an "energy ladder," from less-expensive and less-
5) convenient fuels to more-expensive and more-convenient fuels.
6)
7) In the script below, the "convenience" of fuels is a numerical
8) parameter. In practice, this might come from an expert panel or
9) from a household survey.
10)
11) NOTE: The parameters in the equations below are to illustrate the
12) IPAT-S language only: they are not estimated from data.
13) :comment
14)
15) baseyear 2000
16) scen years 2005 to 2020 by 5
17)
18) dim household 'low' 'mid' 'high'
19) dim fuel 'dung' 'wood' 'coal' 'kerosene' 'LPG' 'electricity'
20)
21) # Declare variables (entries for households are per household)
22) summ var FuelBudget{household} aveFuelBudget
23) summ var totFuelUse{household}
24) var HHShare{household}
25) ratio income{household}
26)
27) summ var FuelUse{household fuel}
28) summ var totFuelByFuel{fuel} totFuel
29) var convenience{fuel} cost{fuel}
30)
31) # Set up hh distribution table -- constant all years
32) HHShare{household = 'low'} = 20%
33) HHShare{household = 'mid'} = 60%
34) HHShare{household = 'high'} = 20%
35)
36) # Initialize values for households

```

```

37) FuelBudget.by{household = 'low'} = 50
38) FuelBudget.by{household = 'mid'} = 150
39) FuelBudget.by{household = 'high'} = 750
40)
41) totFuelUse.by{household = 'low'} = 105
42) totFuelUse.by{household = 'mid'} = 130
43) totFuelUse.by{household = 'high'} = 190
44)
45) income{household = 'low'} = gr(<1.50%>)
46) income{household = 'mid'} = gr(<2.25%>)
47) income{household = 'high'} = gr(<3.0%>)
48)
49) # Calculate future household characteristics -- use elasticities
50) :: >> income^<0.75> -> FuelBudget
51) :: >> income^<0.25> -> totFuelUse
52)
53) summarize FuelBudget * HHShare as aveFuelBudget
54)
55) report aveFuelBudget as "Ave. HH fuel budget (l.c./hh)"
56)
57) # Set up convenience/cost parameters
58) convenience{fuel = 'dung'} = 1
59) convenience{fuel = 'wood'} = 2
60) convenience{fuel = 'coal'} = 3
61) convenience{fuel = 'kerosene'} = 4
62) convenience{fuel = 'LPG'} = 5
63) convenience{fuel = 'electricity'} = 6
64)
65) cost{fuel = 'dung'} = 0.2
66) cost{fuel = 'wood'} = 0.5
67) cost{fuel = 'coal'} = 1.2
68) cost{fuel = 'kerosene'} = 2.5
69) cost{fuel = 'LPG'} = 4.2
70) cost{fuel = 'electricity'} = 7.0
71)
72) # Next, solve the LP for each year and see how fuel use changes
73) LP:
74) solve for FuelUse
75) max convenience * FuelUse
76)
77) cost * FuelUse <= FuelBudget
78) FuelUse = totFuelUse
79)
80) :LP
81)
82) :: HHShare * FuelUse -> totFuelByFuel -> totFuel
83) report 100 * totFuelByFuel/totFuel as "Fuel share (%)"

```

Nearly all of the features of this script should be familiar from the previous chapter. However, taken all at once can be a challenge. In this chapter the script will be analyzed piece by piece. The other sample scripts distributed with IPAT-S can be analyzed in a similar way.

Lines 1-13: The comment block

Lines 1-13 of the script contain an extended comment block.

```
1) comment:
```

```

2) This script implements a model inspired by the "energy ladder"
3) hypothesis. The hypothesis is that households at different income
4) levels climb an "energy ladder," from less-expensive and less-
5) convenient fuels to more-expensive and more-convenient fuels.
6)
7) In the script below, the "convenience" of fuels is a numerical
8) parameter. In practice, this might come from an expert panel or
9) from a household survey.
10)
11) NOTE: The parameters in the equations below are to illustrate the
12) IPAT-S language only: they are not estimated from data.
13) :comment

```

Text in the comment is free-form. It helps to clarify the meaning of the script for people who might read it, and as a reminder for the original script developer of his or her intentions.

Lines 15-16: Year declaration

The script itself starts with a year declaration, as any IPAT-S script must. The declaration should look vaguely familiar, but introduces a shorthand syntax for labeling base and scenario years.

```

15) baseyear 2000
16) scen years 2005 to 2020 by 5

```

Note that `base year` does not have to be two words – it can be written as `baseyear`. This is also true of `scenario years`, which can be written `scenarioyears`, `scenyears` or `scen years`, as it is here.

Lines 18-19: Dimension declarations

Dimensions are used in this script, so following the rules of IPAT-S syntax, the year declaration is followed by dimension declarations. There are two dimensions, `household` and `fuel`.

```

18) dim household 'low' 'mid' 'high'
19) dim fuel 'dung' 'wood' 'coal' 'kerosene' 'LPG' 'electricity'

```

Note that in this example the shorthand `dim` is used instead of `dimension`.

Lines 21-29: Variable declarations

Following the year and dimension declarations, there is considerable freedom in specifying the rest of the script. For ease of maintenance and to make the script more legible, the author of this script chose to group all of the variable declarations together in one place, but the only requirement in IPAT-S is that a variable be declared before it is used. Sometimes it can be useful to defer variable declarations until later in a script – for example, if there is a minor intermediate variable that it is not important for a casual reader of the script to know about.

```

21) # Declare variables (entries for households are per household)
22) summ var FuelBudget{household} aveFuelBudget
23) summ var totFuelUse{household}

```

```

24) var HHShare{household}
25) ratio income{household}
26)
27) summ var FuelUse{household fuel}
28) summ var totFuelByFuel{fuel} totFuel
29) var convenience{fuel} cost{fuel}

```

There are several points to note about this group of declarations. First, the declarations are preceded by an inline comment saying that the variables will be declared. A clarifying comment is added saying that values for households are for a single household rather than, for example, a national total across all households. Second, shorthand is used for most of the declarations – for example, `summ var` rather than `summable variable` and `var` rather than `variable`. Third, most variables have dimensions – either `household`, `fuel`, or both. Fourth, and finally, there are examples of summable variables, variables and ratios in this sample script.

Lines 31-47 and 57-70: Initializing variables

Once variables have been declared, it is usually necessary for them to be initialized, either for a single year or multiple years. In this script, there are initializations in two parts of the code.

```

31) # Set up hh distribution table -- constant all years
32) HHShare{household = 'low'} = 20%
33) HHShare{household = 'mid'} = 60%
34) HHShare{household = 'high'} = 20%
35)
36) # Initialize values for households
37) FuelBudget.by{household = 'low'} = 50
38) FuelBudget.by{household = 'mid'} = 150
39) FuelBudget.by{household = 'high'} = 750
40)
41) totFuelUse.by{household = 'low'} = 105
42) totFuelUse.by{household = 'mid'} = 130
43) totFuelUse.by{household = 'high'} = 190
44)
45) income{household = 'low'} = gr(<1.50%>)
46) income{household = 'mid'} = gr(<2.25%>)
47) income{household = 'high'} = gr(<3.0%>)

```

and

```

57) # Set up convenience/cost parameters
58) convenience{fuel = 'dung'} = 1
59) convenience{fuel = 'wood'} = 2
60) convenience{fuel = 'coal'} = 3
61) convenience{fuel = 'kerosene'} = 4
62) convenience{fuel = 'LPG'} = 5
63) convenience{fuel = 'electricity'} = 6
64)
65) cost{fuel = 'dung'} = 0.2
66) cost{fuel = 'wood'} = 0.5
67) cost{fuel = 'coal'} = 1.2
68) cost{fuel = 'kerosene'} = 2.5
69) cost{fuel = 'LPG'} = 4.2
70) cost{fuel = 'electricity'} = 7.0

```

In this sample script, there is a mix of declaration styles. From lines 31-34, the `HHShare` variable is initialized. But since this is a definition that remains the same for all years, no year is specified in the declaration; in this case, by default, the value on the right of the equals sign is assigned to all years. The variables `convenience` and `cost` are also assigned a value that does not change over the scenario, in lines 58-63 and 65-70. There is no reason these shouldn't change over the scenario, but in this case the author of the script chose to keep them at their base-year values.

From lines 36-43, values are assigned to the base year only, as indicated by the `.by` following the variable names. Scenario values will be assigned to these variables later in the script.

Finally, in lines 45-47, values are assigned for income growth rates by income group. These are key inputs, so they are put in angle brackets. Also, since they are ratio variables, they specify how things change in the future, so by their nature they apply to the scenarios, rather than to the base year. Only a single value is assigned to each income group. By default, the single value is applied to all scenario periods.

Lines 49-55 and 72-83: Calculating and reporting indicators

The actual scenario calculations take place in a few lines. This is one of the strengths of IPAT-S: calculations are compact and expressive, making it easy to create and maintain scripts with a little practice. The bulk of most scripts is devoted to variable initialization, which is hard to make short.⁶

The script calculations and report statements (indicator declarations) occur in two blocks in the script.

```
49) # Calculate future household characteristics -- use elasticities
50) :: >> income^<0.75> -> FuelBudget
51) :: >> income^<0.25> -> totFuelUse
52)
53) summarize FuelBudget * HHShare as aveFuelBudget
54)
55) report aveFuelBudget as "Ave. HH fuel budget (l.c./hh)"
```

and

```
72) # Next, solve the LP for each year and see how fuel use changes
73) LP:
74) solve for FuelUse
75) max convenience * FuelUse
76)
77) cost * FuelUse <= FuelBudget
78) FuelUse = totFuelUse
79)
80) :LP
81)
82) :: HHShare * FuelUse -> totFuelByFuel -> totFuel
83) report 100 * totFuelByFuel/totFuel as "Fuel share (%)"
```

⁶ The IPAT Data eXchange tool is designed to make this job easier, by providing a spreadsheet-like interface for data entry. The data can then be exported to an IPAT-S format.

In lines 50 and 51, fuel budgets (in local currency, “l.c.”) and total fuel use (in unspecified energy units) are driven by income raised to the power of an elasticity. The elasticity is a key input, so it is put in angle brackets. The expressions are applied by the IPAT-S interpreter to households at all three income levels by default.

In line 53, the household fuel budgets are multiplied by the share of the population for each household type and summed over household type using the `summarize ... as statement`. The statement

```
summarize expression as total
```

is a synonym for

```
:: expression -> total
```

It can be used for clarity when writing a script. Anyone reading the script later is likely to interpret the `summarize ... as statement` correctly, that it is summing over some dimension in an expression or variable. The alternative expression may not be as clear.

Line 55 reports the result of the calculation. This could be done at this point, or all report statements could be grouped at the end. There are advantages to each approach, and either is acceptable in IPAT-S.

Lines 73-80 are really the heart of the script – they implement a linear program that maximizes “convenience” while ensuring that each household type stays within its budget and meets its energy needs. Note that dimensions are not specified in these expressions. If there is a mismatch of dimensions on either side of a constraint in the LP, the IPAT-S carries out the appropriate calculations. For example, in line 78, `FuelUse` is automatically summed over all fuels to give `totFuelUse` for each household category.

Line 82 uses a chain expression to calculate two variables in one expression: `totFuelByFuel` and `totFuel`. As `totFuelByFuel` is calculated, the IPAT-S interpreter sums over all household types, while for `totFuel` it sums over all fuels. With these variables it is possible to calculate fuel shares by fuel, which is then reported in line 83.⁷

3.2 Discussion

An IPAT-S script of eighty-three lines is long enough to carry out some fairly sophisticated calculations. The script might be daunting if the language is unfamiliar, but hopefully this example has shown that breaking a script down line by line can make it easier to understand.

The IPAT Studio installation includes several sample scripts that illustrate most (but not all) of the features of IPAT-S. As with any programming language, browsing the samples can be

⁷ This script was written before the IPAT-S syntax `var{dim = sum}` and `var{dim = ave}` were introduced. They might provide a better way of calculating this indicator.

a useful way to learn about the language. Also, they can be a useful starting point for new scripts: if an existing script does part of what you want for a script of your own, then you can either copy and paste the relevant lines or make a copy of the script and edit it until it does what you want.

4 Back of the Envelope Plus

IPAT-S can be used for a range of project sizes, from quick one-off studies to large projects. Since this book is an introduction to the language, it will focus on the smaller end of the scale. This chapter looks at how IPAT-S and IPAT Studio can be used as an enhanced “back of the envelope”.

There are advantages to using software for quick studies, rather than an actual envelope. First, software is easier to share; second, it is easier to return to later; third, it is easy to extend if an idea leads in more complex directions than first envisioned. However, these beneficial results must be balanced against the inconveniences of using a programming language rather than just sketching out some ideas on a napkin or the back of an actual envelope. IPAT-S and IPAT Studio have been designed to minimize the inconveniences. It is relatively straightforward to develop quick studies in IPAT-S and then extend them if that is desired.

4.1 Exhausting Fuel Resources

To demonstrate how IPAT-S and IPAT Studio can be used as a “back of the envelope plus,” an example will be taken from John Harte’s classic textbook, *Consider a Spherical Cow: A Course in Environmental Problem Solving*.⁸ In his problem, “Exhausting Fuel Resources (I),” Prof. Harte asks, “At the 1980 global consumption rate of petroleum, how long will it take to use up the estimated worldwide resource of this fuel?” Conveniently for us, his question specifies the base year (1980). The calculation is meant to be carried out for a single year, rather than over a scenario – below, the question will be modified to embed it in a scenario.

From the appendix in *Consider a Spherical Cow* we find that in 1980 1.0×10^{22} Joules of oil remain and that in that year 1.35×10^{20} Joules were consumed, which provides us all the information we need to write a basic script.

The basic script

The information provided in *Consider a Spherical Cow* can be put quickly into an IPAT-S script:

```
base year 1980

summvar Resources AnnConsumpt

Resources = 1.0e22           # Joules
AnnConsumpt = 1.35e20       # Joules per year

report Resources/AnnConsumpt as "Years to go"
```

There are no scenario years declared because the problem is given in terms of a historical base year, 1980. In IPAT Studio, the script looks like the following.

⁸ Published in 1988 by University Science Books in Sausalito, California. Prof. Harte has since published a second volume, *Consider a Cylindrical Cow*. The titles refer to a joke about how physicists think about the world.

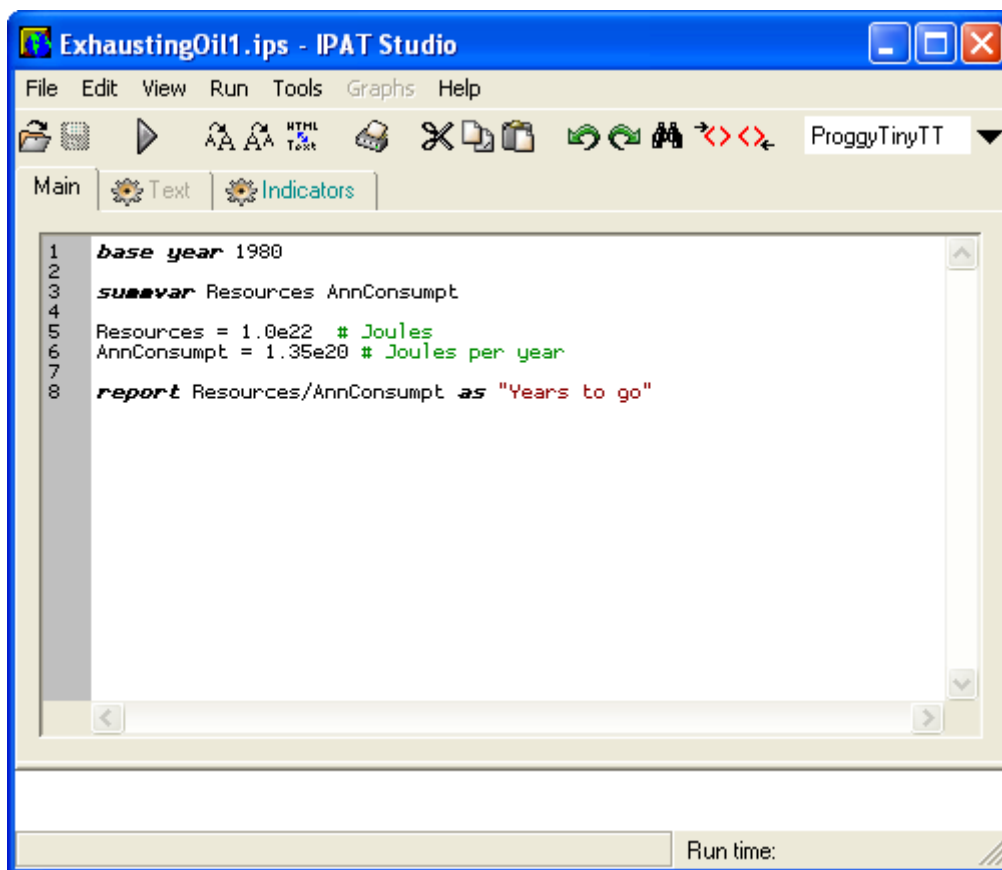


Illustration 2: A script in IPAT Studio

The Indicators tab is active because there is a report statement in the script. The HTML tab is inactive because there are no print statements. Clicking on the Indicators tab runs the script and shows the output. No values are graphed because there is only one data point.

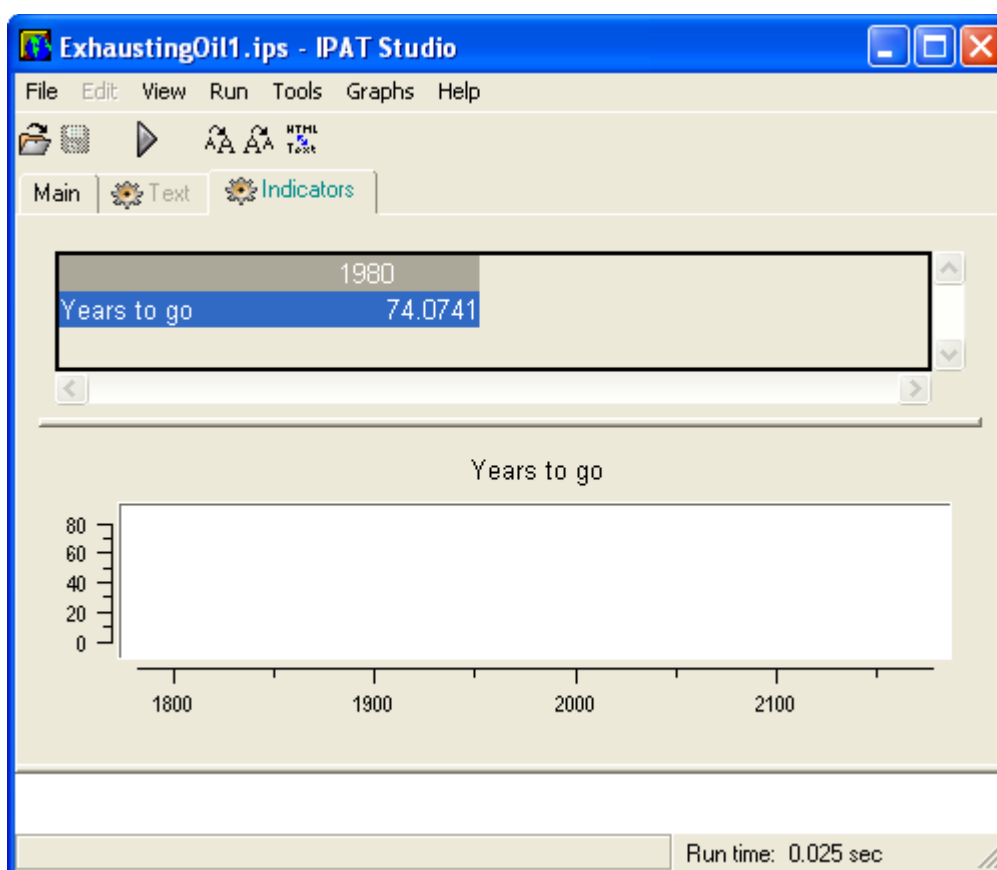


Illustration 3: Viewing indicators in IPAT Studio

Enhancing the script

Because nothing is changing over time, there is only information for the historical year of 1980: in that year there were an estimated 74 years to go before resources would be exhausted *if* consumption rates remained at 1980 levels. However, the value will certainly change over time. It would be interesting to see what it was in 2000, a more recent historical year, and then estimate what it *could* be in 2020. The years-to-go will change over time for a couple of reasons. First, consumption, represented by `AnnConsumpt`, will gradually deplete the available resources, and cumulative consumption must be subtracted from resources before determining the number of years to go in any given year. For this purpose, the IPAT-S function `accumulate()` can be used. This function integrates over the (linearly interpolated) rate given in its argument. Also, it is very likely that annual oil consumption will change over time.

Some possible factors contributing to changing rates of oil consumption are:

- Growing populations.
- Increasing consumption per person.
- Increasing energy efficiency.
- Shifting toward or away from oil as a fuel source.

Different assumptions regarding these parameters can lead to different future rates of oil consumption. At a coarse-grained level, future consumption can be expressed as

$$\text{Oil Consumption} = \text{Population} \times \text{Income} \times \text{Energy Intensity} \times \text{Oil Share}$$

Each of these factors is subject to uncertainty, but population is the least uncertain. Global population in 1980 was about 4.5 billion people; in 2000 it was around 6.1 billion and in 2020 is expected to be around 7.1 billion. These values can be entered directly into the IPAT-S script.

The share of oil in total fuel consumption in 1980 was around 46%, while by 2000 it was around 40%, and had been stable at close to that value for about 15 years. These values can also be entered explicitly in the script.

Changes in income and energy intensity can be expressed as growth rates. Historically, intensity declined at about 1.41% per year from 1980 to 2000, while income grew at about 1.51% per year over the same period (both values using GDP at purchasing power parity).⁹

Assuming the oil share to remain at its 2000 value and assuming growth rates in income and intensity continue into the future, the script becomes

```
base year 1980
scenario years 2000 2020

summvar Resources AnnConsumpt
summvar Pop
var OilShare
ratio income intensity

Pop = 4.5, 6.1, 7.1           # Billions
OilShare = 46%, <40%>

income = gr(<1.51%>)
intensity = gr(<-1.41%>)

Resources = 1.0e22           # Joules
AnnConsumpt = 1.35e20       # Joules per year

:: Pop >>income * intensity * OilShare -> AnnConsumpt

report (Resources - accumulate(AnnConsumpt))/AnnConsumpt \
      as "Years to go"
```

In this script, `AnnConsumpt` is driven by `Pop`, moderated by changes in other factors – income, (energy) intensity and `OilShare`. Income is assumed to rise, while aggregate energy intensity is assumed to fall. The accumulated annual consumption gradually depletes resources, which is implemented by subtracting `accumulate (AnnConsumpt)` from `Resources`.

With the new script, the *Years to go* indicator gradually declines.

⁹ Using readily-available online data from the U.S. Department of Energy (<http://www.eia.doe.gov/>) and the UN Environment Program's GEO data portal (<http://gridca.grid.unep.ch/geoportal/>).

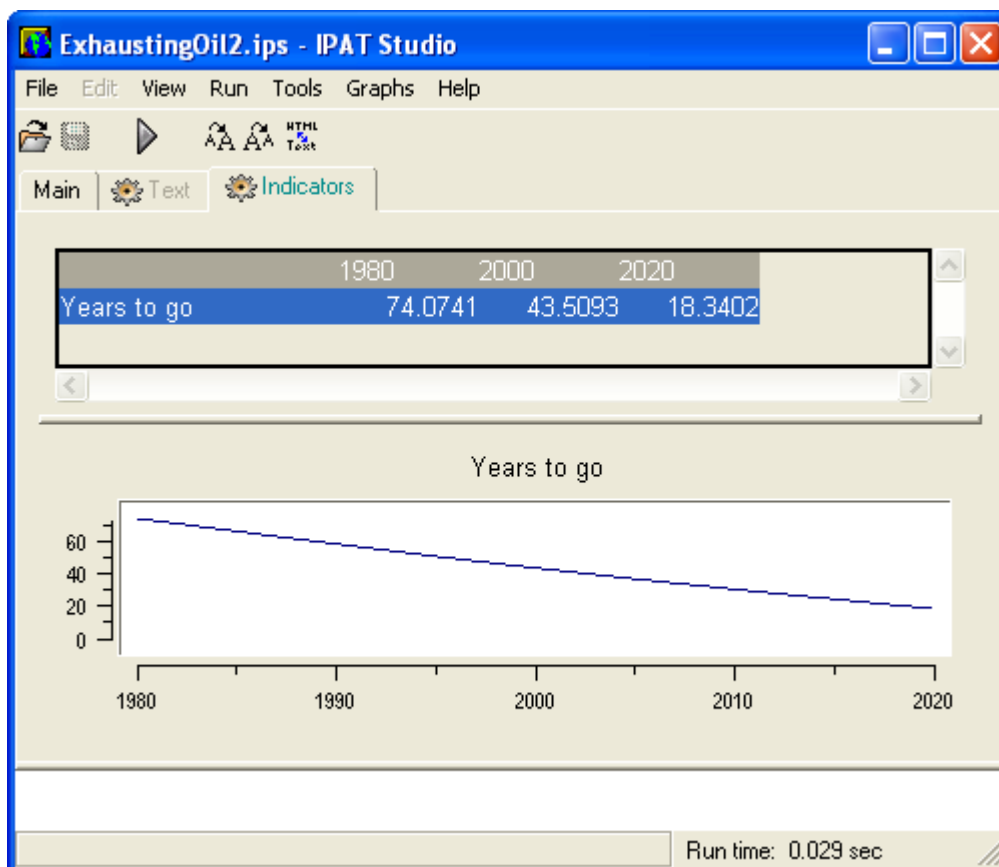


Illustration 4: Second run of the script

However, the assumptions made for income, energy intensity, and oil share over the 2000–2020 period might be either optimistic or pessimistic. There are two ways we might proceed. We could allow the user to interactively change the input assumptions or we could create several scenarios with different assumptions. For this example, let us take the second approach.

To introduce scenarios, we must create a `scenario` dimension. Let us create four scenarios. In each of them, population and income growth are the same. However, in two of the scenarios, energy intensity decreases rapidly, while in the other two it decreases more slowly. (We could have scenarios in which intensity increases, but aggregate energy intensity has tended to decline historically.) Also, in two of the scenarios oil as a share of total energy consumption increases, while in the other two it decreases. The four scenarios can then be given descriptive names:

- **Deep in a Hole:** slowly declining intensity, rising oil share
- **Many Holes:** slowly declining intensity, declining oil share
- **Shallow Hole:** rapidly declining intensity, rising oil share
- **On Solid Ground:** rapidly declining intensity, declining oil share

With these names, consumption of energy resources is represented by digging “holes.” The least disruptive approach, “On Solid Ground,” features a reduced dependence on

oil as well as a reduced dependence on energy in general. As with many scenario exercises, the different scenarios are not neutral – from the point of view of energy conservation, the fourth scenario is definitely preferred, although it may not be easy to achieve without significant social and economic impacts.

The new scenario dimension needs to be added to the script. Also, any variable that could differ between scenarios should be labeled by the scenario dimension. With these changes, the first few lines of the script become

```
base year 1980
scenario years 2000 2020

dim scenario 'Deep in a Hole' 'Many Holes' 'Shallow Hole' \
            'On Solid Ground'

summvar Resources AnnConsumpt{scenario}
summvar Pop
var OilShare{scenario}
ratio income intensity{scenario}
```

Next, the values for the energy intensity and oil share need to be entered for each scenario. The calculations themselves do not need to change, because the IPAT-S interpreter takes care of dimensions automatically. The full script is now

```
base year 1980
scenario years 2000 2020

dim scenario 'Deep in a Hole' 'Many Holes' 'Shallow Hole' \
            'On Solid Ground'

summvar Resources AnnConsumpt{scenario}
summvar Pop
var OilShare{scenario}
ratio income intensity{scenario}

Pop = 4.5, 6.1, 7.1           # Billions
OilShare{scenario = "Deep in a Hole"} = 46%, <40%, 50%>
OilShare{scenario = "Many Holes"} = 46%, <40%, 30%>
OilShare{scenario = "Shallow Hole"} = 46%, <40%, 50%>
OilShare{scenario = "On Solid Ground"} = 46%, <40%, 30%>

income = gr(<1.51%>)
intensity{scenario = "Deep in a Hole"} = gr(<-1.41%, -1.00%>)
intensity{scenario = "Many Holes"} = gr(<-1.41%, -1.00%>)
intensity{scenario = "Shallow Hole"} = gr(<-1.41%, -3.50%>)
intensity{scenario = "On Solid Ground"} = gr(<-1.41%, -3.50%>)

Resources = 1.0e22           # Joules
AnnConsumpt = 1.35e20       # Joules per year

:: Pop >>income * intensity * OilShare -> AnnConsumpt

report (Resources - accumulate(AnnConsumpt))/AnnConsumpt \
      as "Years to go"
```

The results are interesting, as shown in the following graph, which was exported from IPAT Studio.

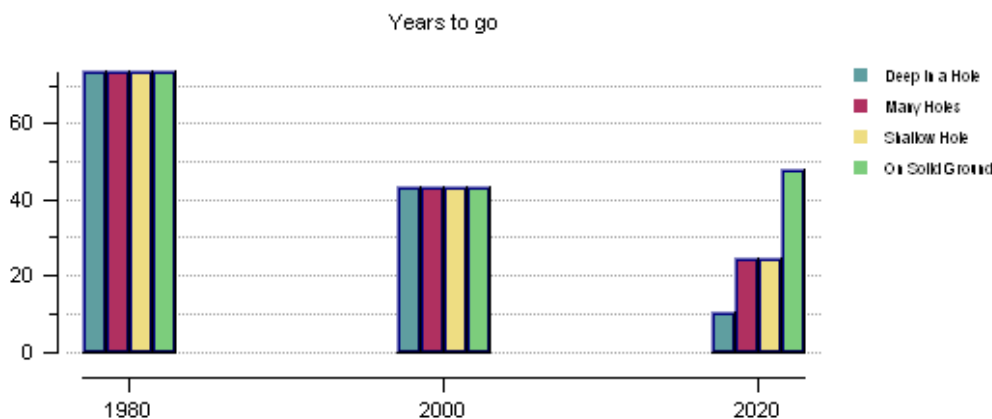


Illustration 5. Indicator graph exported from IPAT Studio

In the “Deep in a Hole” scenario, 10 years of oil reserves are left by 2020 and in both the “Many Holes” and “Shallow Hole” scenarios they become dangerously low. Between “Many Holes” and “Shallow Hole” the trade-off between declining intensity and changing fuel share can be seen. Finally, in the “On Solid Ground” scenario the combination of reduced intensity and declining oil share leads to a small *increase* in the years to go between 2000 and 2020 – although reserves are continuing to be depleted, the rate of depletion is slowing to the point where the remaining reserves can be used for a longer time.

4.2 Discussion

Some of the advantages of using the IPAT Studio environment over a physical back of an envelope can be seen in this simple example. It was easy to create the initial calculation – not much more difficult than doing the calculation on a calculator or in a spreadsheet. But it was also easy to extend the problem, examining possible changes over time and the depletion of reserves. The simple tabular output and graphing capabilities of IPAT Studio made this easy to do. As new questions arise – What happens to carbon emissions? What about other energy resources? What about new oil discoveries? Does disaggregating energy consumption change the picture? What if energy consumption follows an inverted-U, Kuznets curve shape? – they can be implemented by modifying the existing script.

5 Recipes

A programming language must please many people carrying out a variety of activities. In the case of IPAT-S, it must satisfy the people who develop scripts, people who run them and people who read and maintain them. To support these different audiences, the IPAT-S syntax has been designed to support particular kinds of calculations and tasks. In making a language easy for one purpose, it is inevitable that it will be somewhat awkward for other purposes.

Over time, people who use a language get used to its peculiarities and discover the best way to go about solving problems that the language was not specifically designed for. They also discover better ways to do what the language *was* designed for. That is, they discover *recipes* – step-by-step ways to solve common problems. This chapter will present several recipes.

5.1 IPAT and ImPACT

There are several patterns for sustainable development problems. One of the most enduring is the IPAT framework. “IPAT” is short for a formula,

$$\text{Impact} = \text{Population} \times \text{Affluence} \times \text{Technology}$$

The IPAT approach addresses the task of estimating environmental impacts as a sort of *Fermi Problem*. The physicist Enrico Fermi was famous for asking his students questions for which they could not possibly know the answer. His technique for solving them was a back-of-the-envelope series of calculations that involved breaking the seemingly impossible problems down into workable chunks. The most famous example is the problem of determining how many piano tuners there are in Chicago, which can be estimated by first estimating the share of the population that owns a piano and how often they might need their pianos tuned and then multiplying the estimated shares by the population of Chicago. The IPAT formula (or the ImPACT formula) is a solution to a Fermi problem that might be written, “How will environmental impacts change in the future?” Summarizing the discussion in the 1970s, Commoner, Ehrlich and Holdren suggested in a series of papers that the problem could best be solved by breaking it down into the more manageable intermediate factors of population, affluence and technology.

As suggested by its name, IPAT-S is designed specifically to deal with IPAT-style problems. In an IPAT-S script in which `Population` is specified as a summable variable, `Affluence` and `Technology` as ratios and `Impact` as a summable variable (for example, carbon dioxide emissions), then the IPAT-S expression that implements the IPAT calculation is

```
:: Population >> Affluence * Technology -> Impact
```

The chain expression is designed specifically to work with IPAT-type expressions. Also, the chain expression extends IPAT in three important ways. First, it does not limit the list of variables to population, affluence, technology and impact. Second, where possible, variables can be specified as growth rates (via ratio variables) rather than absolute values (as summable variables or variables). This can simplify data entry and calculations. Third, and finally, the variables in the expression above can have dimensions. That is, IPAT-S supports

a disaggregated IPAT equation in which population might be broken into population groups (e.g., by income group, or urban and rural), technology and impact by sector (industry, agriculture, services), and so on.

Since IPAT was first proposed in the 1970s it has been due for revision, and in the 1990s Paul Waggoner and Jesse Ausubel, of Rockefeller University's Program for the Human Environment, proposed what they called a "renovated IPAT identity," the *ImPACT identity*.¹⁰ It is intended to remedy the shortcomings of IPAT as a conceptual framework for analyzing sustainability issues. The rest of this section explains how the ImPACT identity can be implemented in IPAT-S.

The basic ImPACT calculation is

$$Im = P \times A \times C \times T,$$

where Im represents the impact, P represents population, A represents affluence, C represents consumption, and T represents technology. The change relative to the original IPAT equation is the introduction of the consumption factor. The ImPACT identity can be translated directly into IPAT-S as

```
:: P >> A * C * T -> Im
```

Waggoner and Ausubel distinguish what they call *forces* – the factors of population, affluence, consumption and technology – and the growth rates for those forces. The ImPACT relation can be expressed in terms of growth rates using the IPAT-S syntax

```
:: >> growth(p + a + c + t) -> Im
```

Alternatively, since variable names are case-sensitive in IPAT-S, the ImPACT identity could be represented indirectly in terms of growth rates, using

```
:: >> gr(p) -> P
A = gr(a)
C = gr(c)
T = gr(t)

:: P >> A * C * T -> Im
```

In this script fragment, P is a summable variable, A, C and T are ratio variables, and p, a, c and t are number variables.

In Waggoner and Ausubel's terminology, the combination of C and T represent key *sustainability levers*, which counter the combined *sustainability challenge* of P and A. This can be made explicit in an IPAT-S script by writing

```
var SustChallenge = 100 # The SustChallenge variable is an index
ratio SustLever = C * T

:: P >>A-> SustChallenge >>SustLever-> Im
```

10 Waggoner and Ausubel, "A framework for sustainability science: A renovated IPAT identity," *Proceedings of the National Academy of Sciences* vol. 99 no. 12, pp. 7860-7865, 2002. Available online at <http://phe.rockefeller.edu/ImPACT/>.

Again using Waggoner and Ausubel's terminology, a decline in C is termed *dematerialization*, while declining T represents increasing *efficiency*. This can be made explicit in IPAT-S by writing

```
var SustChallenge = 100 # The SustChallenge variable is an index
ratio SustLever = growth(-(dematerialization + efficiency))

:: P >>A-> SustChallenge >>SustLever-> Im
```

or

```
var SustChallenge = 100 # The SustChallenge variable is an index
ratio SustLever = 1/growth(dematerialization + efficiency)

:: P >>A-> SustChallenge >>SustLever-> Im
```

In these expressions, `dematerialization` and `efficiency` are number variables.

Note that each of the examples above demonstrates a different way of representing the same basic ImpACT calculation. Different ways of writing the IPAT-S script suggest different ways of thinking about agency and the challenge of sustainability. The expressions can be modified and extended if there are known relations between the forces. For example, if consumption is related to affluence via an elasticity b , then the basic ImpACT identity becomes

$$Im = P \times A \times A^{b-1} \times T$$

In IPAT-S, this can be written

```
:: P >> A * A^(b-1) * T -> Im,
```

where b is a number variable.

If P also changes with an income elasticity of b_P and T with an elasticity of b_T , then the corresponding IPAT-S calculation can be written

```
:: A^(bP + 1 + (b-1) + bT) >>-> Im
```

In this example, b , b_P and b_T can be either number variables or ordinary IPAT-S variables. If they are defined as variables, then they can change over time and vary along different dimensions.

A further extension might be to include sectoral detail in a calculation. Here is a complete script that implements a scenario of water use at a sectoral level of detail:

```
baseyear 2000
scenyears 2010 to 2050 by 10

dimension sector 'Agr' 'Mfg' 'Dom'

summvar P I{sector} Itot
ratio A C{sector} T{sector}

P.0 = 283 # million people
# These are water withdrawals in 2000
Itot.0 = 469.00 # cubic km
I.0{sector = 'Agr'} = 46% * Itot.0
```

```

I.0{sector = 'Mfg'} = 42% * Itot.0
I.0{sector = 'Dom'} = 12% * Itot.0

# For scenario, assume same rates as for 1970-1995 period from
# Waggoner and Ausubel. 2002. "A framework for sustainability
# science: A renovated IPAT identity". PNAS 99(12), pp. 7860-7865.

num p = <1.0%> # Annual population growth
:: >> growth(p) -> P

num a = <1.5%> # Income growth rate
A = growth(a) # Income growth rate

# Dematerialization
C{sector = 'Agr'} = growth(<-0.7%>)
C{sector = 'Mfg'} = growth(<-0.6%>)
C{sector = 'Dom'} = growth(<0.0%>) # Figure not mentioned in ImPACT
paper

# Efficiency
T{sector = 'Agr'} = growth(<-1.7%>)
T{sector = 'Mfg'} = growth(<-5.0%>)
T{sector = 'Dom'} = growth(<-2.0%>) # Figure not mentioned in ImPACT
paper

# The ImPACT formula:
# - Population drives changes in impact
# - Modified by income, dematerialization and technology
:: P >> A * C * T -> I

report 100 * I/I.2000 as "Water use by sector (2000 = 100)"

summarize I as Itot

report 100 * Itot/Itot.2000 as "Total water use (2000 = 100)"

```

Note that in this script the expression for the ImPACT formula is the same as in the aggregated version. This feature of IPAT-S makes it relatively easy to disaggregate an aggregated analysis. In most cases, the expressions for the main scenario calculations stay the same, since dimensions are handled implicitly.

Running this script and viewing the output in the IPAT Studio, it can be seen that while agricultural and public water withdrawals increase modestly over the scenario, manufacturing withdrawals drop precipitously, leading total withdrawals to decline over the scenario, as shown in the figure below.

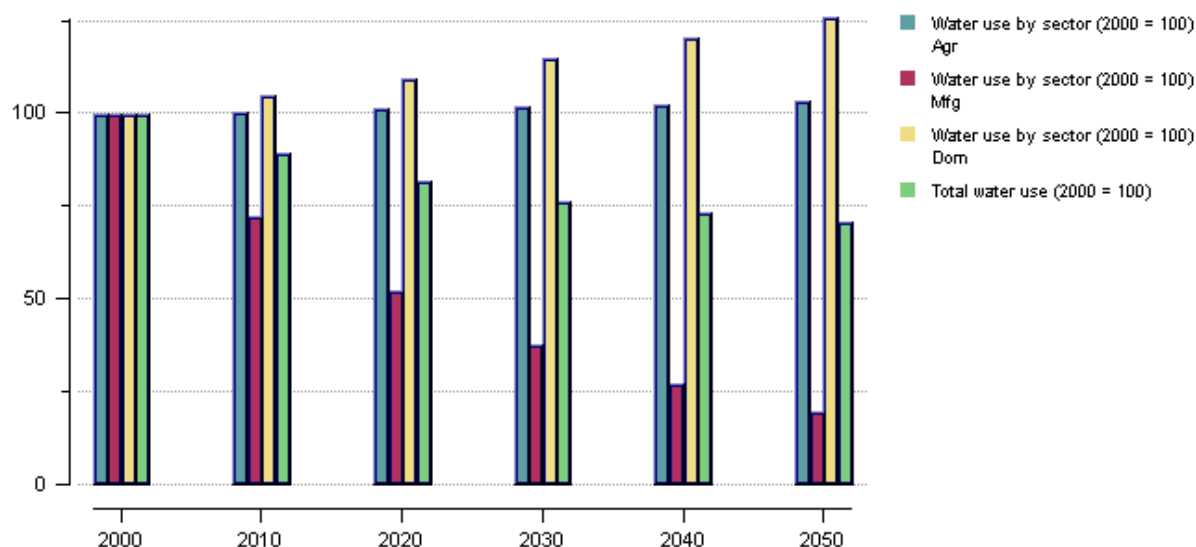


Illustration 6. Output from a disaggregated IMPACT calculation

5.2 Explicit Time Dependence

In most chain expressions, time dependence is implicit rather than explicit. For example, in the following script fragment,

```
Pop = 523, 537, 542
:: Pop >> Income -> GDP
```

the change over time in population is given explicitly, but GDP is calculated using population and an income growth rate (stored in the `Income` variable). Sometimes it is useful to put an explicit time dependence directly into the script. This can be done in more than one way.

Time-dependent chain expressions

One way to implement an explicit time dependence is by constructing time-dependent chain expressions that use the built-in `year` variable, which can also be written as `y` or `yr`. For example,

```
:: Vinit + (Vfinal - Vinit) * (y - y.0)/(y.fin - y.0) -> V
```

will interpolate between a value of `Vinit` in the initial year, `y.0`, and a value of `Vfinal` in the final year, `y.fin`. The `year` variable can be used anywhere a variable or summable variable can be used (including in coefficients for linear programs), and returns the value for the year.

An alternative to a linear time dependence is an exponential dependence, such as

```
:: Vinit + (Vfinal - Vinit) * (1 - exp(-r * (y - y.0))) -> V
```

or

```
:: Vinit + (Vfinal - Vinit) * (1 - (1 - r) ^ (y - y.0)) -> V
```

In these expressions, r is a rate (in % per year). Over time, V gets closer to the value V_{final} without ever reaching it (that is, V_{final} is an asymptote for V).

Logistic curves and “loglets”

Frequently, variables change over time by growing slowly at first, then rising quickly, and finally growing slowly as they approach their maximum. For example, the adoption of a new device might start slowly as a few bold individuals begin to use it, followed by a rapid burst of adoption as it becomes popular, followed by a gradual decline in adoption as the market becomes saturated. This general pattern is characteristic of the *logistic curve*.

Researchers at Rockefeller University’s Program for the Human Environment (PHE) have been using logistic curves to describe many phenomena. Also, they have explored what happens when logistic curves are added together. This is likely to be a common occurrence, because although advances in the use of one technology might saturate, further technologies are likely to come along, so the actual trajectory over time looks like one logistic curve overlaying another. The PHE researchers call these “loglets.”

Loglets can be implemented in IPAT-S. Support is provided through a procedure called `_Loglet` in an external library called `IPATS_Standard.dll`, which is included in the IPAT-S installation. Also, PHE offers LogletLab, software for decomposing time-series data into a sequence of logistic curves.¹¹ Here is an example script using loglets that is available in the standard IPAT-S distribution:

```
comment:
The "loglet" procedure in the IPAT-S standard library
implements a time series as a sum of logistic curves
(a "loglet" decomposition).

For information on the loglet approach, and to
download the LogletLab software, go to
http://phe.rockefeller.edu/LogletLab/.
:comment

base year 2000
scenario years 2010 2020 2030

load _Loglet from 'IPATS_standard.dll' as Loglet

summvar x

#
# To use the Loglet procedure, specify for each term:
#
# max value, characteristic duration (in years), midpoint (year)
#
# An arbitrary number of terms can be introduced
#
```

¹¹ <http://phe.rockefeller.edu/LogletLab/>

```

call Loglet using x \
                100      25      2015 \
                150      15      2025 \
                30       5       2020

report x as "x"

```

The script demonstrates how to load a procedure from an external library (a DLL), as well as the specific way to call the loglet library. The `_Loglet` procedure (renamed to `Loglet` in this script) constructs a sum of logistic curves. Each curve is characterized by three parameters, the same three parameters returned by the `LogletLab` software: the maximum value, the characteristic duration in years and the midpoint year. (The characteristic duration is the time it takes for the curve to go from 10% of the maximum value to 90% of the maximum.)

Note that arguments to external procedures cannot have dimensions.

5.3 Intensities

A common approach to developing a scenario is to start with a variable that sets an overall scale of activity, such as GDP, and then use that to drive resource use and pollutant emissions using an *intensity*, such as the aggregate energy intensity (the ratio of energy use to GDP). In this case,

$$\text{Energy Use} = \text{GDP} \times \text{Energy Intensity}$$

By specifying how GDP and energy intensity change over time, energy use in the scenario is determined. This is the approach that motivates the IPAT framework, and IPAT-S is designed specifically to work with this type of calculation.

A question that often arises is, how should the intensities in a scenario be determined? There are many possibilities. Four will be discussed here – the use of *benchmarks*, *income elasticities*, *Kuznets Curves* and *S-shaped curves* – as well as some hybrid approaches that combine the three.

Benchmarks

The easiest way to develop a scenario for an intensity is to move toward a benchmark. For example, suppose that in a scenario for per capita car travel in a country a part of the scenario narrative is that per capita travel will approach the current value for Denmark. This idea of approaching a set value over time is the same pattern seen in Section 5.2, Explicit Time Dependence (page 31). It can be implemented in the same way, using an expression like

```

:: byv(TravPerCap) + (TravPerCap_Denmark - byv(TravPerCap)) \
  * (y - y.0)/(y.fin - y.0) -> TravPerCap

```

This will linearly interpolate the variable `TravPerCap` between the base year value and the level `TravPerCap_Denmark`. Alternatively, an exponential time dependence could be used, such as

```
:: byv(TravPerCap) + (TravPerCap_Denmark - byv(TravPerCap)) \
  * (1 - exp(-r * (y - y.0))) -> TravPerCap
```

or

```
:: byv(TravPerCap) + (TravPerCap_Denmark - byv(TravPerCap)) \
  * (1 - (1 - r) ^ (y - y.0)) -> TravPerCap
```

Finally, intensities can change over time following a logistic curve, as described in Section 5.2.

Income and price elasticities

Increasing income is often accompanied by increases in consumption, but typically consumption per capita does not rise at the same rate as income. In fact, for some items (such as dung for fuel), consumption per capita could decrease as incomes rise. Similarly, a rise in price is usually accompanied by a decline in consumption.¹²

This can be implemented in a straightforward way using an *income* or *price elasticity*. An elasticity of ϵ (the Greek letter epsilon) means that a 1% increase in income or price is accompanied by an ϵ % increase (or decrease) in per capita consumption. If ϵ is greater than one, then consumption increases faster than income. If it is less than one it increases more slowly than income. If ϵ is zero, then consumption doesn't change at all as income or price increases (it is *inelastic*) and if it is negative, then consumption decreases as income or price increases (for income, it is called an *inferior good*).

This can be implemented easily in IPAT-S using a statement like¹³

```
:: >> income ^ e_inc * price ^ e_price -> Intensity
```

or

```
:: income ^ e_inc * price ^ e_price >>-> Intensity
```

In the first version, `income` and `price` are ratio variables and `e_inc`, `e_price` are numbers. In the second, `income`, `price`, `e_inc`, and `e_price` are all variables. The second variant should be used if the elasticities change over time or across dimensions.

Using intensities can often be a simple and straightforward way to introduce a time dependence into a scenario.

As an example, suppose that a small island nation is interested in how rising airline prices might affect its medium-term tourism revenues. A 1990 World Bank working

¹² But not always! Some goods (called “positional goods”) are valued *because* they are expensive, and can be used to signal the owner’s position in society. Also, in some cases a price is used to signal that a commodity has value.

¹³ This expression is equivalent to saying that a 1% increase in income leads to an ϵ % change in the intensity for very small changes in income and intensity. Proving it requires calculus, but using it doesn’t!

paper by Oum, Waters and Yong estimates that price elasticities for vacation travel by air most likely range from -1.10 to -2.70. Assuming that the share of global tourist arrivals by the island does not change as prices change, then changes in per capita tourism could be estimated using expressions something like

```
# Elasticities from Oum, Waters and Yong, 1990, A Survey
# of Recent Estimates of Price Elasticities of Demand for
# Transport, WB WPS 359
e_low = <-1.10>
e_high = <-2.70>
e_ave = 0.5 * (e_low + e_high)
:: >> baseIndex * price ^ e_low -> Trav{scenario = "Low Impact"}
:: >> baseIndex * price ^ e_ave -> Trav{scenario = "Moderate"}
:: >> baseIndex * price ^ e_high -> Trav{scenario = "Big Impact"}
```

Note that a comment is used to give the source of the estimates directly in the script, so anyone reading the script later can track the source down. Also, in addition to the price impact on travel, a `baseIndex` is also applied, capturing any non-price effects (better advertising, the opening of a new park, income-driven change, the impact of climate change or pollutants on ecosystems, etc.). The `baseIndex` variable might also change with the scenario or be the same in all scenarios, depending on how it was defined earlier in the script.

Kuznets Curves

For many situations, a simple, constant elasticity is useful. However, in some situations consumption or pollution first increases with income and then decreases. For example, for a household using dung for fuel, as their income increases they are likely to turn to more convenient fuels, such as charcoal. However, as their income increases even more, their charcoal use will decline as they use more kerosene. In this case the intensity of coal use is said to follow an “inverted-U” shape, or a “Kuznets curve.”¹⁴

There is more than one way to construct a function with an inverted-U behavior. Two will be shown here, in which a variable x (for example, emissions per capita) changes with income per capita, y . As a general rule,

- Use the first formulation if an inverted-U behavior is expected, but there is little information to go on.
- Use the second formulation if a source gives separate low-income and high-income elasticities, where the low-income elasticity is positive and the high-income elasticity is negative.

Kuznets I

One common form that appears in the literature is

¹⁴ Kuznets proposed an inverted-U shape for the curve of income inequality plotted against income. It turns out that Kuznets’ hypothesis about inequality is not well supported by data, but inverted-U shapes have been spotted in other contexts, and in honor of the famous economist, they are called “Kuznets curves.”

$$x = x_m e^{-k(\ln(y/y_m))^2} .$$

Using this formula, x reaches its maximum value, x_m , when income y reaches y_m . The elasticity ϵ of x with respect to y changes continuously,

$$\epsilon = -2k \ln(y/y_m) .$$

To pin down a value for k , a typical elasticity for, say, $y = 1/2 y_m$ might be specified. Calling this value $\epsilon_{1/2}$, k can be solved as

$$k = \frac{\epsilon_{1/2}}{2 \ln 2} .$$

This version of a Kuznets curve can be implemented fairly easily in an IPAT-S script. In the following script, rather than specifying x_m explicitly, the inverted-U formula is used to create an index that drives the variable.

For the example, x will be replaced by an IPAT-S variable, `IntensityPerCap`, and y will be replaced by an IPAT-S variable `Income`.

```
base year 2000
scenario years 2010 2020

number MaxIncome = <10000>
number ElastHalf = <1.25>
number k = ElastHalf / (2 * ln(2))

var IntensityPerCap, Income
IntensityPerCap.0 = 100
Income.0 = 7000

:: >>gr(<2.0%>)-> Income

:: exp(-k * (ln(Income/MaxIncome))^2) >>-> IntensityPerCap

report Income as "Income ($/cap)"
report IntensityPerCap as "Intensity per capita"
```

Kuznets II

The first version of a Kuznets-like inverted-U curve is convenient and quick. Also, it uses very few parameters. However, for some problems it has too *few* parameters. Also, the elasticity changes strongly at all income levels. It is sometimes convenient to assume a low-income regime with one elasticity, a high-income regime with another elasticity, and then use the Kuznets curve to match the two regimes in the middle. The second version of the Kuznets curve has this behavior.

The second form for the Kuznets curve obeys the equation

$$x = \frac{x_m (\epsilon_1 + \epsilon_2) (y/y_m)^{\epsilon_1}}{\epsilon_2 + \epsilon_1 (y/y_m)^{\epsilon_1 + \epsilon_2}} .$$

The per capita intensity x hits its maximum value x_m at income y_m . The formula is more complicated than the first formulation, but it has a nice behavior:

- When y is much smaller than y_m , x has an elasticity of approximately ε_1 .
- When y is much larger than y_m , x has an elasticity of approximately $-\varepsilon_2$.

This makes it very easy to use this formula when a source for elasticities gives a separate low-income and high-income elasticity, as long as the low-income elasticity is positive and the high-income elasticity is negative.

The following script demonstrates the use of the second form for the Kuznets curve. It is adapted from the example script for the first form, but because the functional forms are different, they give different results.

```

base year 2000
scenario years 2010 2020

number MaxIncome = <10000>
number LowIncElast = <1.25>
number HighIncElast = -<0.75>

var IntensityPerCap, Income
IntensityPerCap.0 = 100
Income.0 = 7000

:: >>gr(<2.0%>)-> Income

# Define intermediate variables for convenience
var z
number e1 e2
:: Income/MaxIncome -> z
e1 = LowIncElast
e2 = -HighIncElast

:: ((e1 + e2) * z^e1)/(e2 + e1 * z^(e1 + e2)) \
    >>-> IntensityPerCap

report Income as "Income ($/cap)"
report IntensityPerCap as "Intensity per capita"

```

Asymptotic curves

In some cases, an intensity may start rising at low incomes with some elasticity, but then eventually saturate. For example, it can be expected that the share of household energy use from electricity eventually saturates. In this case, an *asymptotic* curve – one that gradually approaches a limiting value – can be a good choice for a modeling function.

Typically, in this case, data will be fit using a *logit* specification. In this approach, the variable x that is saturating is transformed using a logit transformation, and then regressed on income:

$$\ln\left(\frac{x}{x_m - x}\right) = a \ln y + b \quad .$$

If the regression coefficients are then rewritten so that they match the form

$$\ln\left(\frac{x}{x_m - x}\right) = \epsilon \ln(y/y_{1/2}) \quad ,$$

by setting

$$\begin{aligned} a &= \epsilon \\ b &= -\epsilon \ln y_{1/2} \quad ' \end{aligned}$$

then the formula for x is

$$x = \frac{x_m (y/y_{1/2})^\epsilon}{1 + (y/y_{1/2})^\epsilon} \quad .$$

This has some nice features:

- When y is much smaller than $y_{1/2}$, x has an elasticity of approximately ϵ .
- When y is much larger than $y_{1/2}$, x asymptotically approaches x_m .
- When y is equal to $y_{1/2}$, x is equal to $1/2 x_m$.

This formula is implemented in the following script.

```
base year 2000
scenario years 2010 2020

number HalfIncome = <10000>
number Elast = <1.25>

var IntensityPerCap, Income
IntensityPerCap = 100
Income.0 = 7000

:: >>gr(<2.0%>)-> Income

# Define intermediate variable for convenience
var z
:: Income/HalfIncome -> z

:: z^Elast / (1 + z^Elast) >>-> IntensityPerCap

report Income as "Income ($/cap)"
report IntensityPerCap as "Intensity per capita"
```