LEARNING EXERCISES 1

1.1. Introductory level

The first exercises with GAMS are intended to familiarise you with the basic structure of GAMS and the IDE interface. The goal is that you are able to write, run and read GAMS models and their output.

Exercise 1.1.1. Starting GAMS-IDE

The first exercise should be easy:

Start the GAMS interface program GAMS-IDE by clicking on the GAMS-IDE button (the icon says IDE in red and has two black squares below the name: ).
Go to the right directory and make a new (or open an existing) project.

If there have not been created any projects on this machine before, you'll get a window asking you to provide a project or name a new project. Go to your own directory[1] and make a new project (*e.g.* my project.gpr).

If GAMS-IDE opens with a different project with open files, close any files that are open by selecting "File / Close" from then menu, or by clicking the small **x** in the right-hand corner of the window containing the file. Warning: do not press the small **x** in the upper right-hand corner, as this will close GAMS-IDE and not just the file. Now you can make a new project (or open the existing project).

Actually, you can switch between projects without closing your files first. In this case next time you open your project these files are automatically opened.

The next step is to create a new file by selecting "File / New" from the menu. Save this empty file by selecting "File / Save" from the menu, or by pressing the Save button () on the toolbar. Give the file the name "intro".

The program will automatically add the extension ".gms". You could use other names, but for the practical it is useful that you stick to the names suggested here.

Exercise 1.1.2. The basic blocks of writing a model in the IDE

If necessary, open the file "intro.gms" in the project you chose in Exercise 1.1.1. Type the simple model as presented at the end of Section 1.5. (see Reader 1) Write all the basic blocks of code (parameters, variables, equations, model, solve). Carefully check for typing and other errors.

[1] Note: in some cases there have been problems with using a network drive as the project directory (especially when using Windows 2000). In those cases, create a directory on the hard disk.

Exercise 1.1.3. Reading the process window

Run this GAMS file by selecting “File / Run” from the menu, by pressing the F9-key or by clicking on the button on the toolbar ().

The .gms-file is automatically saved before running it.

A process window appears. This window displays the log of the model run: it tells you what GAMS is doing, it shows information on the iterations of the solver and in this window you can read whether GAMS has solved the model correctly. Please take a careful look at the completion status of the model: if the process window informs you (at the bottom) that the model has ‘normal completion’ this does not necessarily mean that an optimal solution was found. So for each solve, carefully check that an optimal solution was found.

Depending on the settings of the project, the process window automatically moves to the last line written (it automatically scrolls down when GAMS moves on), or it stays at the top of the log. In the first case, you can manually scroll back up to look at earlier information, and in the latter case you can manually scroll down. (You can change this setting by selecting “File / Options” from the menu, then checking “Update process window” from the “execute” tab).

Exercise 1.1.4. Reading the listing file

When you run a GAMS file, a so-called listing file is automatically written by GAMS. So when you ran the file intro.gms, GAMS made the file intro.lst in the directory where the project is located (i.e. in the working directory).

Note: do not mix up the log file and the listing file. The log file is what GAMS-IDE shows during the run; the listing file is where GAMS stores the results of your model run.

The .lst output file can be activated using the Open command on the File menu. However, it is usually easier to first survey a run by examining the log file in the separate Process Window, which is automatically displayed. A brief log of the run appears there, and clicking on any of the black-coloured lines (including run error messages) will activate the entire .lst output file and position you on that message. In particular, clicking on Reading solution for model will open the .lst and position the window at the SOLVE SUMMARY. Clicking on a red-coloured line will cause the cursor to jump in the .gms file to the line with the error.

Open the listing file intro.lst.

Read the output file carefully by following each item of the discussion of the general structure of a GAMS output file below.

The general structure of a GAMS output file is[2]:

1. Echo print.

The first part of a list file gives a copy of the GAMS input file that has been run. For the sake of future reference (in case of errors) GAMS puts line numbers on the left hand side of the list file.

[2] Note that parts of the output file may look different if you use a different solver.

2. Error messages. {Only available if there are mistakes in the model!}

If you have made any errors then these are displayed with a dollar sign ($) followed by a number. The error number and an explanation of the error is given after the echo print. If any errors have been made, then these should be corrected before proceeding. Errors are the topic of the next exercises.

3. Reference maps.

The next section of the list file is a pair of reference maps that contain summaries and analyses of the input file for the purposes of debugging and documentation. The first reference map is a cross-reference map such as one finds in most modern compilers. It is an alphabetical, cross-referenced list of all the identifiers (e.g. scalars/parameters, variables, equations) of the model. The second reference map is a list of model identifiers grouped by type and listed with their associated documentary text.

4. Equation listing.

If no errors have been made then the list file will also contain an equation listing. The equations listing allows you to check whether GAMS has indeed generated the model from your input file that you intended.

5. Model Statistics.

This is the last section of output that GAMS produces before invoking the solver is a group of statistics about the model's size.

6. Solve summary.

After the solver executes, GAMS prints out a brief status report. **It is important to <u>always</u> check the Solver Status and the Model Status to verify that GAMS has found the optimal solution**.

7. Solution reports.

This is the part that you are of course most interested in. The solution report of the variables gives the results of the optimisation problem that was solved in GAMS. The report presents lower, level, upper and marginal values, where lower and upper will give any lower and upper bounds that have been imposed. Level gives the optimal value for the variable, i.e. the solution. The value under marginal gives the dual value (shadow price) of the variable; this is the first derivative of the objective value to the level of the variable.

Read the solution report of the file intro. What are the optimal values, lower and upper bounds of the variables?

8. Report summary.

At the end of the solution report, a report summary is given. The desired report summary should be as follows :

```
**** REPORT SUMMARY :    0     NONOPT
                         0     INFEASIBLE
                         0     UNBOUNDED
                         0     ERRORS
```

If you have made any errors, then correct them now.

Exercise 1.1.5. Debugging a compilation error

If you have carefully carried out the exercises above, you have not seen any errors yet. So in this exercise, we will make one intentionally.

Remove the semi-colon after equation `QX1`. Run the model again and see what the process window tells you.

Double-click the red line with the first error in the process window. You automatically jump to the file intro.gms and are near the point where the error is (in this case, you'll jump to the line below the error). Double-click on the error message in the process window, just below the red line. The listing file appears and the error code is shown. More below in the listing file, you'll find an explanation of the error codes, so you can check what type of error you've made.

Fix the error and save the correct file.

(If only you were always so lucky to know what the error is straight away.)

Trying to fix any errors you get when building a model is called *debugging*. This debugging is often the hardest and most time consuming part of the whole process of model building. Through learning-by-doing you'll get better at debugging as you get more experience. So don't feel bad if it takes you forever to fix a simple error when you've just started specifying GAMS model.

Exercise 1.1.6. Debugging a logical error

A second type of errors commonly made can be labelled as 'logical errors'. These errors are not a conflict with the GAMS syntax, but a misformulation of the model itself. These logical errors are sometimes hard to find, especially if you do not fully understand the logic behind the model. And sometimes you do not even get an error, but you get just the wrong results. Therefore, always think hard about the specification of the model and make sure the specification is alright.

In this exercise we'll find out what happens if we want to minimise `Y` instead of maximising it.

Replace '`MAXIMIZE`' with '`MINIMIZE`' in the `SOLVE` statement. Explain why the model fails to get an optimal solution. Look at the listing file to find out what the model status is. Repair the error and save the correct file.

Exercise 1.1.7. Building a new model

Now that you have learned to build and debug a GAMS model, you should be able to implement your first model in GAMS. Let's start with a general description of the model:

Due to the existence of open access pastures there exists a tragedy of the commons. In west Africa pasture productivity is threatened by overgrazing, a headtax is envisioned to induce farmers to lower their livestock numbers. If we consider an area of 1000 hectares the present number of cattle is 2000 (without a headtax). The number of cattle supply decreases with 50 for each currency unit of tax imposed. The stocking rate is defined as the number of cattle per hectare. The damage per hectare due to grazing is the square of the stocking rate and valued at cost of 24 currency units. Livestock keeping itself is seen as a social

beneficiary activity and valued at 50 currency units per head of cattle. Taxation is seen as undesirable. Hence, the social objective function consists of the positive effects of cattle keeping minus the tax burden and minus the value of damage due to overgrazing.

Implement this model in a new GAMS file called “intro2.gms”.

Note: if the model is solved properly, it will give a MODEL STATUS 2: locally optimal. This does not mean that there are 2 local optima. In non-linear models, like this one, GAMS will never give a model status 1: optimal.
Hints:

- you can use the following parameters: `pastures` (=1000), `cattle0` (=2000), `taxcoef`(=–50), `benefit` (=50), `cost` (=24) and `tax` (=30).
- you need the following variables: `damage`, `st_rate`, `cattle`, `obj`. Each variable gets its own equation based on the description above:

```
eq_cat..  cattle =E= cattle0 + taxcoef*tax;
eq_sr..  st_rate =E= cattle/pastures;
eq_dam.. damage =E= cost*st_rate*st_rate*pastures;
eq_obj.. obj =E= (benefit-tax)*cattle – damage;
```

Change the tax rate (`tax`) from 30 to 20. How does this influence the social objective (`obj`)? What is the value for the social objective if the tax rate is 10? And if the tax rate is 0?

Exercise 1.1.8. Understanding what you have done

This exercise is to test whether you understand what you have been doing so far.

Hint: use the information in the appendix to check your model results.

a. Go back to the file intro (if necessary, open the file again). Replace the defined value for variable `X2` in the equation by a parameter `B` that has the same value (`5`). Make all the necessary changes to the model so that the model works and the same values for `X1` and `Y` result. Check whether GAMS has found an optimal solution. Repair any errors you get underway.

b. As a final test at this level, rewrite the model to have an environmental-economic meaning:

Replace equation `QX1` with `QPRD.. PRD =G= 100`;
Replace equation `QY` with `QEMIS.. EMIS =E= CO2+OTHER`;
Add the equation `QCO2.. CO2 =E= coef*PRD`;
Rename the model to `CLIMATE`;
Change the objective from maximizing `Y` to minimizing `EMIS`;
Make sure all parameters and variables are declared and the parameters are given a value (`OTHER=5` and `coef=0.03`) and remove all redundant code.

Run the model and analyse the results.

If you feel confident that you grasp the exercises above, go to the next level. If not, try to play around with the model some more (for example, you can specify parameter `OTHER` as a variable).