Gsim Programming Tutorial

Daniel Romero
University of Agder, Norway
daniel.romero@uia.no
Version: 2018/06/18

Acknowledgements: Luis M. Lopez-Ramos, César Asensio-Marco, Bakht Zaman

GSim is a MATLAB environment to run simulations and real-data experiments that

- **automates repetitive tasks** performed by the researcher, which saves time and reduces the chance of mistakes. Note that mistakes when conducting a simulation study are potentially very harmful for your professional activities.
- lays a **framework for collaboration** among multiple researchers, which enables code reuse.
- facilitates **interaction with processing servers**.

GSim comprises two parts:

- An **execution environment**, which assists the researcher when running experiments on MATLAB. Knowledge of *object-oriented programming* (OOP) is NOT required.
- A **simulation environment**, which structures the code into *processing blocks*. These blocks are reusable by other researchers and enable automatic figure drawing and execution parallelization. Knowledge of OOP IS required.

Both parts can be used separately or jointly. The present document describes the usage of the execution environment solely. Documenting the simulation environment is the subject of ongoing work.

GSim Execution Environment

The execution environment comprises code for helping the researcher conducting a simulation study. Its motivating aspects are:

- It allows the programmer to have **simulation code properly organized** in “experiment” functions. This means that after several months without touching the code, you will not have to figure out which of your (possibly tens or hundreds of) .m files contains the simulation that you are looking for.
- **Store simulation results** in an orderly fashion. You can plot results of a simulation you already run without having to figure out which one of your (possibly hundreds of) .mat files contains the results. Furthermore, with GSim, you will always know which code generated each file of results without having to guess and making mistakes.
- Easily **export any figure to pdf**, which produces pdf files ready to include in a paper. Also straightforward to store figures in .fig format. Fonts are embedded in the resulting pdf, as required by some conferences.
- It facilitates and speeds up **interaction with servers**. After running an experiment, a server will store the resulting figures in a standard fashion that allows one to easily plot, edit, and export it afterwards in the local computer.
- Enables us to benefit from the **features of** the class **GFigure** (see below).

Installation

Copy the folder `GSim` to a directory in your computer. In MATLAB, go to that folder and execute

```
>> gsimStartup
```

in the command line. This command will create a file named `gsim.m`. This file contains the function that you will invoke to execute any experiment.

To check that you have properly installed GSim, execute

```
>> gsim(0)
```

This will display the results of a dummy experiment, where a random matrix is generated and shown on the screen.

Run a Simple Simulation

Experiments are organized in experiment files. These files are located in the folder `Experiments/` and its name typically ends in *Experiments.m*, such as `TemplateExperiments.m` or `TutorialFfigureExperiments.m`. We typically have one experiment file per simulation study, which in most cases corresponds to one conference and/or one journal paper.

The experiment file to be used is selected in `gsim.m`. In a fresh installation, the selected file is `TemplateExperiments.m` by default. Later, we will explain the file `gsim.m` and how experiment files are selected. At this point, let us have a look at `TemplateExperiments.m`.

Open `TemplateExperiments.m` and observe that there is a section of *properties* and a section of *methods*. The section of methods contains one function per experiment, which are named `experiment_N`, where N is a number identifying the experiment and can be any positive integer. In general, to execute the experiment with number N, one types

```
>> gsim(0,N)
```

in the command line. For example, type

```
>> gsim(0,1002)
```

to execute experiment 1002. You will see that a figure has been displayed on the screen.

The code to generate such a figure is inside the function `experiment_1002`. You can see that the first three lines

```
v_x = -10:0.01:10;
v_y = v_x.^2;
plot(v_x,v_y)
```

just plot a quadratic function in the interval [-10,10]. The next line,

```
F = GFigure.captureCurrentFigure();
```

takes the contents of the figure that has been plotted and stores it in the object `F`. Do not worry if you are not familiar with objects; just think of this object as a data structure that stores the current figure inside. The function `experiment_1002` then returns this object `F`.

When you type `gsim(0,1002)` in the command line, the function `experiment_1002` is executed and the object `F` that it returns is stored in your file system, specifically in the folder `Experiments/<name_of_experiment_file>_data/`. Once the experiment has been run, you can plot the stored figure anytime by typing

```
>> gsim(1,1002)
```

In this case, the function `experiment_1002` is not invoked. Note that this behavior is very convenient since it allows the researcher to plot the results of a simulation experiment without running the experiment again. If GSim is not used, one can of course store the results of a simulation in a file, but this approach is slower and people do not tend to be systematic when naming and storing that file. This typically implies that finding the results of an experiment run several days ago may be difficult and prone to errors.

Now, feel free to experiment by modifying the code inside `experiment_1002`. For instance, replace the line

```
v_y = v_x.^2;
```

with

```
v_y = randn(1,length(v_x));
```

and run `gsim(0,1002)` to plot a realization of white noise.

If you want to see how to write an experiment that produces multiple figures, see experiment 1003 in `TemplateExperiments.m`.

The File gsim.m

The file `gsim.m` is the entry point of all experiments. It is used to specify which experiment you want to run and how it must be run. In collaborative repositories, each researcher will have his own version of `gsim.m`. The function in this file does not do any “real job”; the purpose of `gsim.m` is just to avoid the need for writing long statements in the command line.

If you open `gsim.m`, you can see that, after a help header, it comprises four sections:

1. A section with **initializations**. This is not intended to be modified.
2. A section with **execution parameters** containing
    a. the variable `defaultExperimentClassName`, which is used to select the simulation file.
    b. the variable `defaultExperimentIndex`, which is the number of the experiment that is executed when no experiment number is specified through the command line. In other words, if `defaultExperimentIndex=N`, then

`gsim(0)` will execute `experiment_N`. This is convenient when writing experiments since one tend to execute the same experiment a large number of times until it is totally coded. For those cases, it is handy to create a shortcut icon (see the top right corner of the MATLAB window) that executes `gsim(0)` whenever you click on it and set `defaultExperimentIndex` to the experiment under construction.

   c. the variable `defaultNiter`, which will be described later.
   d. the variable `runningOnServer`, which may be used when running GSim on a server.

3. A section with **parameters for graphical representations** that will be explained later.
4. A final section that invokes the functions that do the "real job", i.e., **running the selected experiment**. This section is not intended to be modified.

Create a File of Experiments

When you start a simulation study, you will typically create a new file of experiments. To this end,

1. Create a copy of the file `TemplateExperiments.m` in the folder `Experiments/` and rename it. Preferably, choose a file name that ends with the word Experiments, for example `MyFirstExperiments.m`.
2. Open the file `MyFirstExperiments.m` and replace the word `TemplateExperiments` with `MyFirstExperiments` in the first line of code.
3. In `gsim.m`, replace the line

```
defaultExperimentClassName = 'TemplateExperiments';
```

with

```
defaultExperimentClassName = 'MyFirstExperiments';
```

You are now all set. You can now create experiment functions inside `MyFirstExperiments.m` to run your code. For instance, add the following function in the section of methods:

```
function F = experiment_2002(obj,niter)

            [X,Y,Z] = peaks(25);
            surf(X,Y,Z);

            F = GFigure.captureCurrentFigure();
end
```

and type

```
>> gsim(0,2002)
```

to execute it. Note that if your experiment does not plot any figure, you must add the line

```
F = [];
```

since otherwise there will be an error.

You can remove or modify the sample experiments (i.e., 1001, 1002, 1003, and 1004) at will. Remember to ensure that the number of the experiment is unique within this file.

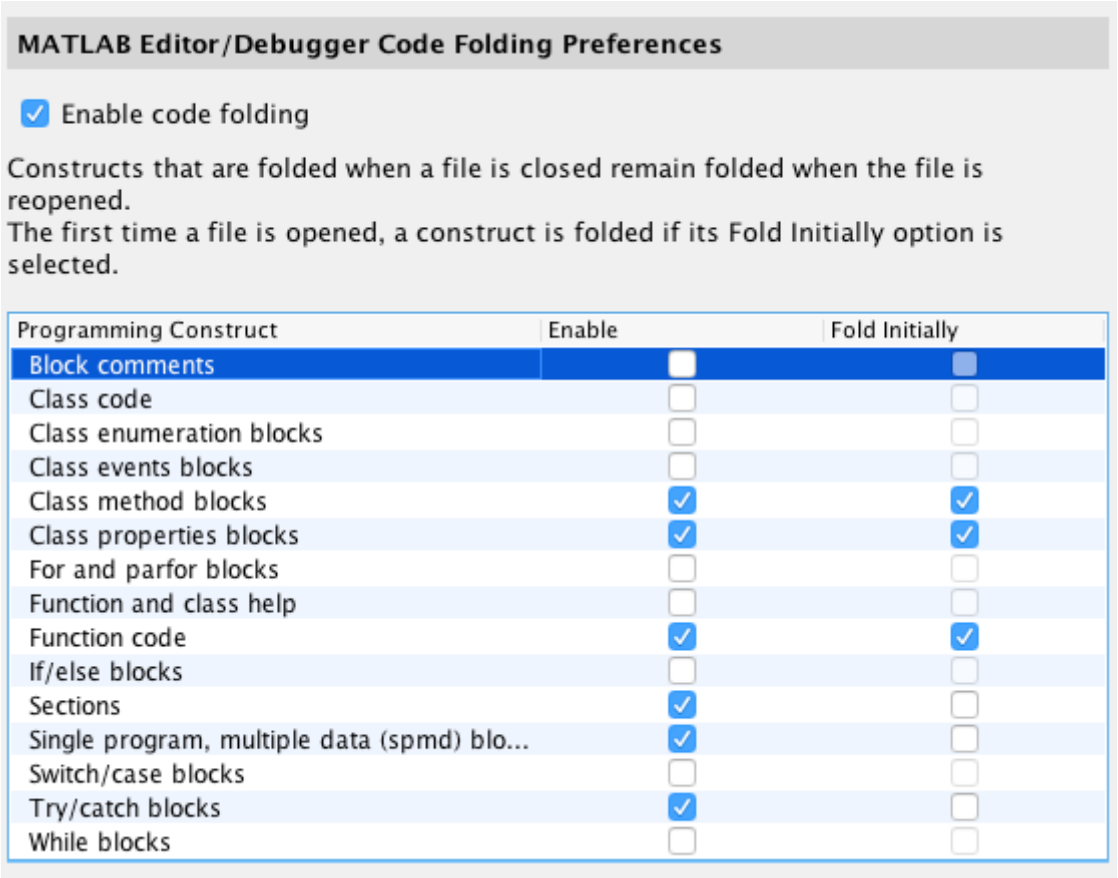
Experiment files constitute a convenient form of organizing experiments. A handy MATLAB functionality is that of *code folding*. With this functionality, one can easily have a high-level view of the experiments, as illustrated in the following figure:

```
classdef TemplateExperiments < ExperimentFunctionSet

    properties
        % You may define here constants that you will use in the
        % experiments of this file

    end

    methods

        % Example of experiment that does not display any figure.
        function F = experiment_1001(obj,niter)...

        % Example of experiment that displays a figure. The figure is
        % first constructed and displayed through MATLAB standard commands.
        % The figure is then stored inside an F_figure object through the
        % function F_figure.getF_figureFromFigure();
        function F = experiment_1002(obj,niter)...

        % This experiment is similar to 1002, but it illustrates how to
        % create more than one figure.
        function F = experiment_1003(obj,niter)...

        % Example of experiment that displays a figure. An F_figure is
        % constructed and returned. GSim will use this F_figure to display
        % a figure.
        function F = experiment_1004(obj,niter)...


    end

end
```

You can even make use of MATLAB sections to group experiments:

```
classdef TutorialFfigureExperiments < ExperimentFunctionSet

    methods

        %% 0. Introduction ====================================================
        %% 1. How to avoid using F_figure =====================================
        %% 2. How to actually use F_figure ====================================
        %% 3. Titles and axes labels ==========================================
        %% 4. Multiple curves =================================================
        %% 5. Subplots and sequences ==========================================
        %% 6. 3D plots ========================================================
        %% 7. Fixing issues without running experiments again =================
        %% 8. Conclusion ======================================================

    end

end
```

A possible configuration that facilitates such high-level views can be set by navigating to `preferences > MATLAB > Editor/debugger > Code folding` and checking the boxes indicated by the following figure.



If you need to use functions or classes in your experiments, two options are recommended:

- Place your code in a folder (preferably not in `Execution/` or `Experiments/`) and add that folder to the path by editing the method `initializePath` in `Execution/initializeGsim.m`, which is invoked every time GSim is executed. You are

encouraged to follow the guidelines on code organization of the project you are working for, if any.

- If the code is very specific to the current simulation study (e.g. a function that tries four methods and returns the MSE for all of them), you may create a function inside the experiment file. To this end, you need to know what is the syntax used in OOP.

Communicating with an Experiment Function from the Command Line

Observe that the experiment functions, such as `F = experiment_2002(obj,niter)` above, have an argument named `niter`. This is a variable that GSim takes from the command line when you invoke `gsim()` with 3 parameters, as in

```
>> gsim(0,<experiment_number>,<niter_value>)
```

The variable `niter` can be used inside your experiment function for any purpose, typically as the number of iterations of an optimization algorithm or Monte Carlo simulation. For example, consider the following simulation:

```
function F = experiment_2003(obj,niter)

     n_randomVariables = 100;
     lambda = 2;
     v_meanEstimate = zeros(1,niter);
     for iter = 1:niter
         v_realization = poissrnd(lambda,1,n_randomVariables);
         v_meanEstimate(iter) = mean(v_realization);
     end
     VarianceOfMeanEstimator = var(v_meanEstimate)

     F = [];
end
```

This experiment uses the Monte Carlo method with `niter` iterations to obtain the variance of the sample mean estimator when it is applied over a vector of 100 realizations to estimate the mean of a Poisson random variable. If you use a low number of iterations, e.g.

```
>> gsim(0,2003,10)
```

you will see that the execution time is short, but the result is not precise, as you can check by executing the aforementioned statement again. However, for a large `niter`, e.g.

```
>> gsim(0,2003,100000)
```

the execution time is longer, but the result is correspondingly more precise.

Observe that when `gsim` is invoked with less than 3 arguments, GSim uses the default value of `niter` specified in `gsim.m`.

The purpose of having a parameter that can be set from the command line is to save time when running code in a server. In a typical situation, a researcher will start by running an experiment with a low number of iterations to estimate how much time it takes to the server to perform each iteration. Depending on that number, he/she can estimate how long it will take to run the experiment with a larger number of iterations. Recall that one should never run a long experiment (e.g. of a few days) without having an approximate idea of how long the experiment will take!! Since `niter` is set from the command line, it speeds up the aforementioned process by avoiding the need for editing source files.

Exporting Figures

Remember that there is a section in `gsim.m` where one can specify graphical representation parameters. The first of these parameters is used to export the figure(s) produced by an experiment function, as required when the figures are to be included in a paper. To this end, set the variable `displaySettings.b_savePlots` to `1` and run (or simply plot the results of) an experiment. For instance, set `defaultExperimentClassName = 'TemplateExperiments'` in `gsim.m` and type

```
>> gsim(1,1003)
```

You will see the following lines on the command window:

```
Saving to Experiments/TemplateExperiments_data/TemplateExperiments_1003-1.pdf
Saving to Experiments/TemplateExperiments_data/TemplateExperiments_1003-2.pdf
```

These are the locations of the pdf files generated. Note that GSim automatically sets the background color to white, as required when writing a paper. MATLAB .fig files are also generated, whose name differ from the above files just on their extension. If your function explicitly creates an `GFigure` object (see next section), a text file with the caption may also be generated.

Note that the appearance of the pdf figure, namely the font size, aspect ratio, and line width, depend on the dimensions of the window that displays the MATLAB figure at the time when it is exported. When preparing a paper, all figures need to have the same appearance, so one needs to ensure that the size of the window is the same when exporting all the figures of the paper. To facilitate this task, the variable `displaySettings.v_windowPosition` in `gsim.m` enables us to specify the dimensions of such a window. One can set this variable to a 1x4 vector and use the same vector when exporting all figures of the paper.

A possible vector that works in some computers is `[100 402 444 273]`. If you cannot see the entire figure on the screen when using this vector, please follow this procedure:

1. Close all figures (`close all`)
2. Set `displaySettings.v_windowPosition` to `[ ]`

3. Plot the results of an experiment that displays only one figure; e.g. by typing `gsim(1,1002)`
4. Using your mouse, resize the window and place it on the desired location of your screen
5. Type

```
>> get(gcf,'position')
```

6. Set `displaySettings.v_windowPosition` in `gsim.m` to the 1 x 4 vector that is printed on the command line.

GFigure

`GFigure` is a class that facilitates and speeds up certain tasks that need to be performed when preparing figures to include in papers. The experiments described so far use the line

```
F = GFigure.captureCurrentFigure();
```

to create an `GFigure` object using the contents of the current displayed figure. The resulting object does not "understand" the contents of the figure, in the sense that it does not know what is the title, legend, curves, and so on.

An alternative to such an *implicit* construction would be to *explicitly* construct the GFigure object by indicating all the information needed to plot the figure. Doing so is not difficult and entails multiple advantages, but it is totally optional. Benefits of explicitly constructing an `GFigure` object include:

- All figures will have the **same format**.
- The resulting **code is more compact**: most of the lines of your experiment code will be devoted to the actual simulation, and only a few to representing the results.
- Certain **default settings** in MATLAB are overwritten to improve the visual quality of the exported figure. For instance, the line width is increased since the default line width of MATLAB is too small for good visibility on printed papers. Also, the set of colors used to display multiple figures improves visibility in papers and slide presentations with respect to the default color set in MATLAB.
- Allows one to **change notation in all figures** of the same paper in a single shot. For example, if a certain variable in your paper is called N and after a certain point you decide to call it K, then GSim+`GFigure` can apply the change by modifying a single line of code in a *translation table*. This is a frequent task, and doing it manually is highly time consuming and error prone.
- Automatically remove the titles from the figures and **produce .txt files containing** the information therein to include in **captions**.

Hands-on Tutorial

The usage of `GFigure` is documented in `GFigure.m`, both in the header and inside the section of properties. A hands-on tutorial is provided in `Experiments/GFigureTutorialExperiments.m`. Just set

```
defaultExperimentClassName = 'TutorialFfigureExperiments';
```

in `gsim.m` and read the file `GFigureTutorialExperiments.m`. There, the usage is illustrated through sample experiments that you can run.

Conclusions

This document described the usage of GSim. The key motivation is to simplify and speed up various daily tasks that researchers perform when conducting a simulation study and writing a paper. Simulations are organized as experiments, and experiments are grouped into sections, and sections into files. GSim also facilitates the preparation of figures ready to include in papers. These functionalities are enhanced by the optional usage of GFigure.

For questions, send an email to daniel.romero@uia.no. Any feedback will also be appreciated.