DSL for the development of interactive tutorials in MATLAB/Octave

line 1: 1st Given Name Surname  
line 2: dept. name of organization  
line 3: name of organization  
line 4: City, Country  
line 5: email address

line 1: 4th Given Name Surname  
line 2: dept. name of organization  
line 3: name of organization  
line 4: City, Country  
line 5: email address line 1: 2nd Given Name Surname  
line 2: dept. name of organization  
line 3: name of organization  
line 4: City, Country  
line 5: email address

line 1: 5th Given Name Surname  
line 2: dept. name of organization  
line 3: name of organization  
line 4: City, Country  
line 5: email address line 1: 3rd Given Name Surname  
line 2: dept. name of organization  
line 3: name of organization  
line 4: City, Country  
line 5: email address

line 1: 6th Given Name Surname  
line 2: dept. name of organization  
line 3: name of organization  
line 4: City, Country  
line 5: email address

*Abstract -* This paper presents a set of tools for creating interactive tutorials in MATLAB and Octave. The tutorials are based on an interactive format, where the student must complete tasks to advance. A Domain Specific Language is used to describe the tasks, which makes it easy for people without programming knowledge to create them. The tools have been tested with teachers and students of different levels of experience, and the results have been positive. Users have reported that the tools are easy to use and that they help them learn more effectively.

*Keywords - Tutorial, Interactive, MATLAB, Octave, DSL*

# Introduction

Unlike other educational documents such as a manual, a tutorial is characterized by offering the student a hands-on experience to develop the skills that are intended to be taught. In this way, it seeks to reach the third level of Bloom’s taxonomy [1], [2], where the student not only understands the subject matter, but also has the ability to apply the acquired knowledge to solve problems in controlled environments.

In this sense, there are two main paradigms in the creation of tutorials:

* **“*Do what I tell you*”**: This consists of presenting explanatory texts with boxes where the student copies and executes the content in a work environment. Platforms such as Comprehensive Rust [3] allow the commands to be executed directly in the electronic document. Other modern examples of this approach are Jupyter notebooks [4] or Google Colab [5].
* **“*Solve what I ask you*”**: It is based on carrying out an assimilation test after the theoretical introduction. For example, a description of Ohm’s law is presented, examples of problem solving and then a battery of questions to evaluate the student's understanding. A prominent example is the Khan Academy [6], with excellent material for learning mathematics.

This work presents a framework for building interactive tutorials in MATLAB/Octave (it has been developed in Octave, but no compatibility problems with MATLAB were found, in this document we will use MATLAB and Octave interchangeably) for teaching subjects that use this environment (the MATLAB/Octave scripting language itself, signal processing, image processing, control, etc.). The proposed framework is based on positive elements of both paradigms.

The proposed tutorial concept consists of a succession of “*experiments*” with a small introductory text, with examples of a concept, and then a list of tasks is proposed to verify that the student has understood the concepts worked on. In this sense, the student can complete freedom to work with MATLAB/Octave to solve the tasks requested. In this way, the user’s work environment is literally MATLAB/Octave integrated development environment (IDE), not a reduced version for the tutorial. Therefore, additionally to the tutorial’s own help, the user has the help provided by the IDE itself (variable inspector, workspace, history, help with hyperlinks, among others). In addition, the framework provides a series of specific commands to help the user in the achievement of the proposed tasks, such as commands to verify that the task is being carried out correctly, construction of lists of clues, glossary of tutorial relevant terms and others.

The system presents a second characteristic that enhances its educational capacity: the construction of the experiments is carried out with a simple own markup language that facilitates the inclusion of contents for the tutorial builder. Specifically, its markup language facilitates the incorporation of explanatory texts, examples to reproduce (from the “*do what I tell you*” paradigm), as well as tasks that the student must solve (from the “*solve what I ask you*” paradigm). Besides, the simplicity of the used markup language enables a very enriching active learning strategy in which students can create their own experiments that are shared with others. In fact, in the pilot test described in this work (see Section IV) were carried out in this sense: the students realize a reduced tutorial of basic instructions, and after a short time with the tool, they themselves will be able to develop new content for their classmates.

To achieve this objective, a transpilation [7], [8] approach has been proposed. Specifically, this transpilation strategy consist of a conversion from a Domain Specific Language (DSL) or markup language to MATLAB/Octave script language. This approach provides two advantages: On the one hand, the tutorial editor does not have to be an expert in MATLAB/Octave. On the other hand, transpilation process adds value to the tutorial without additional cost. In the process of generating the MATLAB/Octave scripts, the system orders the concepts so that they are presented in a natural order (in which the required previous knowledge is presented before the currently worked ones), generation of indexes, preparation of glossaries, clues for each experiment, etc. But this transpilation does not diminish the capacity of the tutorial since octave code can be incorporated directly into the result.

The rest of the work is organized as follows: the description of the tutorial generator from the user’s point of view (command and abilities) is defined in Section II. Section III describes the description of the markup language itself, through which the user writes the experiments in basic form. Later, a pilot experience is described in Section **¡Error! No se encuentra el origen de la referencia.**. Finaly, the conclusions are the Section V.

# Description of the tutorials enviromental use

From the user’s point of view, they directly work in the MATLAB/Octave IDE, in which the following interactive commands have been added:

* ***inicio***. Prepares the system to start the tutorial. It adds the experiment scripts to the path, initializes the current experiment global variable, and loads the first experiment.
* ***ayuda***. Shows help for the commands available in the tutorial system.
* ***indice***. Shows a list of all available experiments.
* ***pista***. Lists the experiments that the current experiment directly depends on. Indicating which previous concepts are necessary to address it.
* ***glosario***. Shows a list of all the terms defined in the tutorial. If it is called with a term ("glosario term"), it shows all the experiments related to that term.
* ***siguiente***. Advances to the next experiment without verifying the current one.
* ***anterior***. Goes back to the previous experiment.
* ***irexperimento(N)***. Jumps to experiment N.
* ***repite***. Restarts the current experiment and shows the description and tasks requested again.
* ***verificar***. Verifies if the tasks of the current experiment have been carried out correctly, in which case it will advance to the next experiment.

Since the users are working directly in the MATLAB/Octave IDE, they have all the advantages that it offers, including the option to autocomplete commands, or use the cursors to retrieve commands from the history. In addition, from an internal point of view, the system consists of a state variable, which indicates the current experiment, and a set of functions that act on this variable and depending on its value.

Each experiment is concentrated in a script file whose general structure is an "if" command that differentiates the code that is executed when the task is being presented and the code that is executed when it is being verified whether the task was carried out correctly or not.

# Description of the Markup Language and development tools

The description of the experiments is carried out on an ad-hoc markup language that is line-oriented. Each line of the text that defines the experiment can start with a tag that defines the purpose of the entire line. If a line does not have a tag, it is considered to be within the explanation text of the experiment. In this way, you have a simple format to interpret (similar to, but even simpler than, markdown [9]). In addition, there are three commands that are interpreted within the text lines of the experiments to highlight words or mark the words that will appear in the glossary.

The Example 1 shows the description (in the markup language) of a very basic experiment. As can be seen, a first block of metadata (identifying the experiment, the author, the dependencies and keywords of the experiment) and then the text that defines the concept and the task to be carried out (the “*q:*” tag puts the following word or number in quotation marks) and finally the verification part in which the variables that the experiment is going to use are initialized and the different verification tests (as many as desired).

The complete list of available tags and commands is provided in Tables I, II, III, and IV. Table I details the metadata tags used to describe the experiment, including information such as subject, author, and dependencies. These tags may have limitations on their cardinality, which refers to the number of times a particular tag can appear within the experiment. The cardinality column in Table I indicates these limitations: “1” signifies that the tag must appear once and only once, while “\*” denotes that it can appear any number of times, including zero.

Tables II through IV categorize and present the remaining tags used in the experiment description based on their specific functions. The last table refers to tags to mark words within the text. And these are placed as a prefix of the word to be tagged using a colon as a separator between the tag and the word (“tag:word”).

In the repository https://github.com/jamarier/octave-kudos is available the full framework (the transpiler, auxiliary tools and some example). Although the platform where the framework was developed is the Linux OS, it is a set of scripts written mainly in Perl 5.x, which has no OS dependencies, so it is compatible with any OS that has a POSIX layer and supports Perl. The utility/library dependencies are:

* Perl modules **IO::All** and **Modern::Perl**
* **graphviz**. To generate the graphical representation of the dependency graphs between the experiments. (Not necessary for the generation of the tutorials).
* **zip.** To compress the final result.
* **gnumake**. To simplify the generation of the tutorials.

In the repository, the originals of the experiments must be in a folder of the “experimentos” directory and in the Makefile file the BLOQUES variable must be adjusted to indicate the directories to use within the experiments.

From there, the following make targets perform the following actions:

* **make build** to generate the MATLAB scripts (which will be in the build directory).
* **make all** to generate the scripts and the graph of the experiments (called “casos.dot.pdf”, see partial example in Fig. 1).
* **make zip** generates a zip file with all the generated code. This generated file is the only thing that needs to be passed to those who are going to use the tutorial, which only requires unzipping it and executing the init command inside the MATLAB/Octave IDE to use it.
* **make zipsrc** compresses the entire project: the Perl scripts and the source codes of the experiments. This command is intended for the second phase in which the students themselves generate new experiments.

# Pilot experience with the tool

Once the operation of the tutorial builder has been described, an analysis of the useful for the students will be carried out. In this sense, this tool was evaluated by the students of the Digital Signal Processing subject of the Degree in Industrial Electronic Engineering at the University of Seville. Specifically, the target of this pilot is facilitating to students the introduction in the MATLAB/Octave environment. The pilot was caried out during the 22/23 academic year, with a total group of 63 students. Thus, since participation was voluntary (receiving extra points in the final evaluation), only 34 students were followed up.

Specifically, the evaluation was carried out in three stages:

In the first, based on a set of description files provided by the teacher, each student self-generated the 54 initial experiments of the tutorial, referring to the basic concept of the Script language studied. With this activity, the student became familiar with the syntaxes of the proposed self-generation environment. Besides, they started the training process with the MATLAB/Octave language itself.

After completing this stage, the students answered a question about the usability of the proposed tool with the results shown in Table V. As can be seen from this evaluation, the students generally considered the proposed tool easy to use, obtaining an acceptance of 3.74 out of 5. Thus, in some comments they stated that it was difficult for them to understand the dynamics in the first files but that, after approximately halfway through the activity, they are already familiar with its use.

In the second phase, each student was challenged to define two description files associated with the set of contests that would continue the tutorial. After the completion of this second activity, the students answered other question about the creating process (see Table VI). In this aspect the proposed tool also obtained a good evaluation with an acceptance of 3.12 out of 5.

Finally, in the third and last stage, the students carried out the experiments proposed by their classmates (66 in total), having concluded the tutorial with a total of 122 experiments.

After completion, the students answered three questions (see Table VII), about the perceived opinion of the proposed tool. In this sense, the evaluation of this final stage maintained a similar trend. Specifically, the interactivity of the tool was highlighting by students, with a score of 3.79 out of 5 in usefulness and 4.09 out of 5 for its capacity for reconfiguration/extension the activities. In this sense, some comments of students explicitly highlighted as “very positive” the possibility to configure or perform as many tests as they wished for the different concepts studied. Finally, the overall rating of the tutorial was 3.53 out of 5.

# Conclusions

The implementation of tutorials provides an excellent opportunity to reinforce knowledge through hands-on learning, because these activities are based on a “*Solve what I ask you*” philosophy. However, the implementation of tutorials poses the challenge of requiring experienced personnel when planning or writing the activities. This in many cases limits the number of activities to be carried out.

In this sense, the present work proposes an academic tool that based on a simple DSL, and through a transpilation process, allows the tutorial users themselves to define practical activities, without requiring knowledge of the subject or the language being addressed.

This powerful has been made available in this work to the teaching community, through a GitHub repository, addressing as example the self-generation of a MATLAB/Octave tutorial.

Finally, this self-generated example has been tested in a real pilot case, during the course 22/23 of Digital Signal Processing of the Escuela Politécnica Superior at the University of Seville.

In this case study, students showed a high degree of satisfaction with the proposed tool, highlighting the usefulness of a tutorial in which they can define *ad-hoc* the number of tests and/or rhythm associated with each concept or idea worked on.

##### References

[1] D. R. Krathwohl, *Taxonomy of educational objectives: the classification of educational goals*. New York: David McKay Company, Inc., 1974.

[2] L. W. Anderson and D. R. Krathwohl, Eds., *A taxonomy for learning, teaching, and assessing: a revision of Bloom’s taxonomy of educational objectives*, Complete ed. New York: Longman, 2001.

[3] “Comprehensive Rust.” [Online]. Available: https://github.com/google/comprehensive-rust

[4] “Jupyter Project.” [Online]. Available: https://jupyter.org/

[5] “Google Colab.” [Online]. Available: https://research.google.com/colaboratory/

[6] “Khan Academy.” [Online]. Available: https://www.khanacademy.org/

[7] B. Wang, A. Kolluri, I. Nikolić, T. Baluta, and P. Saxena, “User-Customizable Transpilation of Scripting Languages,” *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, pp. 201–229, Apr. 2023, doi: 10.1145/3586034.

[8] M. Bysiek, A. Drozd, and S. Matsuoka, “Migrating Legacy Fortran to Python While Retaining Fortran-Level Performance through Transpilation and Type Hints,” in *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*, Salt Lake, UT, USA: IEEE, Nov. 2016, pp. 9–18. doi: 10.1109/PyHPC.2016.006.

[9] “Lightweight Markup Language (Markdown).” [Online]. Available: https://en.wikipedia.org/wiki/Markdown