expCode: beautiful computational experiments

Mathieu Lagrange

July 7, 2015

About you

Let's assume that you will or are practicing computational experiments. Most probably, this takes you time and a lot of care, and may be you have some frustrations depending whether you are:

- **a Master's student ?** Then, you may at some point consider the fact that the problem is not simply to come with a new idea and implement it. To contribute significantly to the research community are striving to be part of, you need to compare your method with the ones of others. This process is tedious, hard if not impossible and involve a lot of coding and knowledge about large scale processing, statistical analysis and reporting of quantitative data.
- **a PhD student ?** You have several years to dedicate to a research project. Doing it well will help you stay motivated and efficient. But how ? Several years of work means a lot of code, a lot of bugs, a lot of failures and hopefully some gain of knowledge for you and your research community. How will you keep track of those many experiments ? How will you efficiently document them ? How will you quickly report your progress to the members of your research team ? How will you publish your research in a reproducible way ?
- **a Post doctoral fellow ?** You are now an established researcher, with many ideas about what could be done in order increase knowledge in your community. But you also have to juggle with many different projects you are involved in. Keeping track of all those projects and being able to easily switch between them is mandatory for success. For example, being able to re-run years old experiments in order to efficiently satisfy a reviewer request is critical for your career.
- **a Full time researcher ?** Besides research, you have many duties that shreds the time you can allocate to pursue the many personal research projects you have. The time needed to switch context is sometimes too long to put you in an efficient research mode for the short time slot you have. More importantly, the free time you have is usually not in front of your desktop. Also, you are advising several students and most of the time, when the student goes away, the project ends at best with a student specific organization of code and data that will most probably not help the next student to pursue efficiently the research project.

And for all or others, you are heading towards sharing your code but you are not confident with your programming expertise and

you do not have time to improve your experimental code into a difusable state[1] ?

If so, please consider giving expCode a try as it is specifically designed with those matters in mind and hopefully will help you reducing specific burden that keep you way from reaching this goal which is one of the most important step towards true expansion of knowledge in science and engineering: reproducibility[2].

[1] http://sciencecodemanifesto.org

[2] R. D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011; and P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research in signal processing. *Signal Processing Magazine, IEEE*, 26(3):37–47, May 2009

Features

expCode is a software framework currently implemented in Matlab that

- provides a high level abstraction of a computational experiment
- allows multiple user per experiment
- allows multiple processing platform to be used
- features an strong decoupling of 3 major experimental phases:
  1. coding
  2. processing
  3. reduction of results.

Benefits

1. The user can focus on solution code
2. evaluated with standard experimental designs
3. bugs and the time dedicated to their solving is reduced
4. context switching between projects is much easier
5. as well as diffusion of reproducible code

The scientific method

The scientific method is a well established method to gain knowledge with demonstrated merit. Sadly, modern ways of doing research impose strong pressure on the time and efforts that can be allocated to a project. The consequence is that important steps of the scientific are often neglected.

We believe that this quest for speed adversely reduces the meanningfulness of the produced results. At the same time, We agree that strictly following the scientific method can be tedious and shortcuts might be tempting. expCode is designed to assist you in the most tedious and less error prone steps and will hopefully help you dedicate more time to the fundamental steps of the scientific method.

Quickly put, the scientific method can be divided into several steps that each may have to be iterated. On the following table, we stated where the expCode framework can be helpful during this iteration process.

The scientific method[3]

Phases	Steps	expCode
Analysis	Describe problem	
	Set performance criteria	
	Investigate related work	
	State objective	
Hypothesis	Specify solution	
	Set goals	
	Define factors	+
	Postulate performance metrics	+
Synthesis	Implement solution	++
	Design experiments	+++
	Conduct experiments	++++
	Reduce results	+++
Validation	Compute performance	++
	Draw conclusions	+
	Prepare documentation	+
	Solicit peer review	

[3] P. Bock and B. Scheibe. *Getting it right: R & D methods for science and engineering*. Academic Press, 2001

Installation

expCode in a nutshell

expCode is designed to provide you with as stream lined set of tools to efficiently build the computational environment you need in order to gain knowledge about a research statement.

For the sake of simplicity, we will now consider a trivial research statement. We want to gain knowledge about the base area and the volume of a few 3 dimensional geometrical shapes.

First, we assume that the expCode framework is in your path, if not, please type the following in your command window:

```
addpath(genpath('<pathToExpCode>'));;
```

Create project

Let us call create the project *geometricShape*: `expCreate('geometricShape');`
The command ends by moving into the experiment directory.

Define steps

The processing steps can be now be instantiated:

```
geometricShape('addStep', 'base');
geometricShape('addStep', 'space');
```

Alternatively, the 3 previous commands can be operated at once:

```
expCreate('geometricShape', {'base', 'space'});
```

Define factors

We are interested in the potential impact of the different attributes (shape, color, radius, width, height) of the geometric shape on its base area and volume. The shape can be a cylinder a pyramid or a cube:

```
geometricShape('addFactor', {'shape', {'cylinder', 'pyramid', 'cube'}});
```

The shape can be blue or red:

```
geometricShape('addFactor', {'color', {'blue', 'red'}});
```

The radius of the cylinder (modality 1 of factor 1) can be 2, 4, or 6 meters:

```
geometricShape('addFactor', {'radius', '[2, 4, 6]', '', '1/1'});
```

The width of the pyramid and the cube (modalities 2 and 3 of factor 1) ranges from 1 to 3 meters:

```
geometricShape('addFactor', {'width', '1:3', '', '1/[2 3]'});
```

The height of the shape is only relevant for processing step 2 and for the cylinder of the pyramid (modalities 2 and 3 of factor 1):

```
geometricShape('addFactor', {'height', '2:2:6', '2', '1/[1 2]'});
```

Factors can be viewed by typing: `<experimentName>();`, thus `geometricShape();` now returns:

```
Factors:
1     shape =  =  = {'cylinder', 'pyramid', 'cube'}
2     color =  =  = {'blue', 'red'}
3     radius =  = 1/1 = [2, 4, 6]
4     width =  = 1/[2 3] = 1:3
5     height = 2 = 1/[1 2] = 2:2:6
```

The factors can be edited in the file `<shortExperimentName>Factors.txt`, that is `geshFactors.txt`.

A setting is a set of modalities, one of each factor of interest.

Implement processing steps

The first step is dedicated to the computation of the base area of the shape. The solution code is implemented in `gesh1base.m`:

```
uncertainty = randn(1, 100);
switch setting.shape
    case 'cylinder'
        baseArea = (pi+uncertainty)*setting.radius^2;
    otherwise
        baseArea  = setting.width^2;
end
store.baseArea = baseArea;
```

We assume here that π is known up to a given precision, but 100 measurements have been made.

The second step build on the result of the first processing step to compute the volume (implemented in `gesh2volume.m`):

```
switch setting.shape
    case 'cube'
       volume = data.baseArea*setting.width;
    case 'cylinder'
        volume = data.baseArea*setting.height;
```

```
    case 'pyramid'
        volume = data.baseArea*setting.height/3;
end
```

Define observations

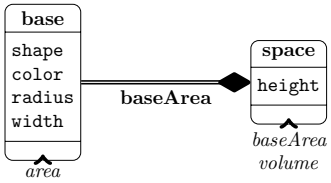
Observations for the 2 processing steps are the following. First step:

```
obs.area = baseArea;
```

Second step:

```
obs.baseArea = data.baseArea;
obs.volume = volume;
```

The command `geometricShape('f')` generates a diagram view of the experiment.



Process

```
geometricShape('do', 1);
```

run the base step over all 18 settings.

```
geometricShape('do', 0, 'mask', {[1 2] 0 1});
```

runs successively every steps over the cylinders of radius 2 and all the pyramids.

Expose observations

Upon completion of the processing, the results of the last processing are displaying in the command window:

This display can be achieved by issuing the following command:

```
geometricShape('display', 2, 'expose', '>', 'mask', {[1 2] 0 1});
```

The exposition of relevant observations is important for efficient computational experimentation. expCode provides many tools for this purpose, type `help expExpose` for quick reference.

The command:

```
geometricShape('display', 2, 'mask', {1 0 1},...
 'expose', {'t', 'obs', 3, 'sort', 1});
```

displays the volume (observation 3) of the step volume (2) for each cylinder of radius 2 sorted according to the first on a table (t). Red color indicates best performance, and blue ones, performances which are statistically equivalent to the best one.

	color	height	volume
1	blue	6	78.35 (27.90)
2	red	6	73.84 (22.61)
3	blue	4	52.23 (18.60)
4	red	4	49.23 (15.07)
5	blue	2	26.12 (9.30)
6	red	2	24.61 (7.54)

For this example, even if the blue cylinder is bigger than the blue one due to the uncertainty in the estimation of π, this difference is not significant, so the blue and red cylinders shall be considered of equivalent volume.

Report

The file `<shortExperimentName>Report.m`, that is `geshReport.m` in this example allows you to generate report that compile several expositions of observations: Latex table (l), bar plot (b), and box plot (x).

```
exposeType = {'l', 'b', 'x'};
for k=1:length(exposeType)
    config = expExpose(config, exposeType{k}, 'obs', 3, ...
        'mask', {1 0 1}, 'save', ['geo' exposeType{k}]);
end
```

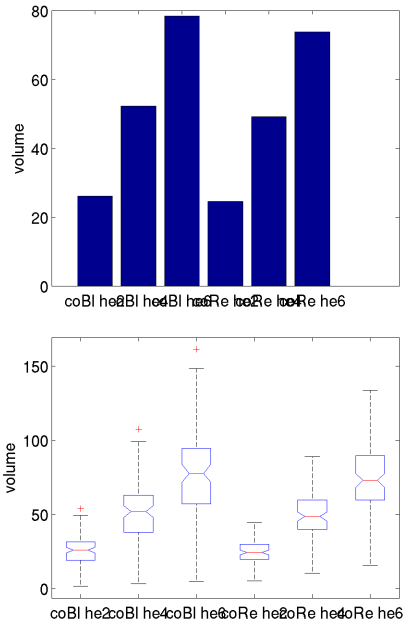
color	height	volume
blue	2	26.12±9.30
blue	4	52.23±18.60
blue	6	**78.35±27.90**
red	2	24.61±7.54
red	4	49.23±15.07
red	6	**73.84±22.61**

Table 1: shape: cylinder, radius: 2

Several reports can be handled for the same experiment using the key `'reportName'`, and if the name of the report contain the key `'Slides'`, a slide presentation layout is used. For example, the command:

```
geometricShape('report', 'rcv', ...
  'reportName', 'presentationSlides');
```

generates a report with base name `presentationSlides` in a slides presentation layout.



Architecture of an experiment

An expCode experiment has a specific directory architecture.

Code

The main directory hosts the main processing routines (named `codePath` in your configuration). It has the following files:

- `<projecName>`: main entry point of the experiment
- `Factors`: text file encoding the factors of the experiment
- `Init`: processed before any prcessing step
- `Steps`: implement each processing units
- `Report`: handle the generation of report that compile several types of expositions of observations

Configuration

The processing environment of the experiment can be controled with the files hosted in the directory `config`.

Almost every processing units of an expCode experiment has access to a structure named `config` which is imported from a file which is specific to each user. This file is created at the creation of the experiment from a user specific file that can be found in the `.expCode` directory in the home directory of the user.

The syntax of this file allows the user to handle several configuration depending on which machine the experiment is processed. For example, let us assume that the project handles 3 machines with different file architectures:

```
machineNames  = {{'toto', 'yoyo'}, 'dodo', 'momo'}
codePath = {'¬/code/geometricShape', '/lab/code/geometricShape}
```

If the experiment is processed on the machine `'toto'`, or `'yoyo'`

Every entry of the configuration file can be overwritten at the command line call. For example:

```
geometricShape('do', 0, 'sendMail', 1)
```

process every steps and sends an email at the end of the processing.

New entries can also be added and accessed in the processing files of the experiment:

```
geometricShape('myEntry', 'toto')
```

outputs:

```
Warning. The command line parameter myEntry is not found
 in the Config file. Setting anyway.
```

Data

Access to input data and processing data are respect

Report

Commands

Management of an experiment

Computation of settings

Exposition of observations

Best practices

factorial design

multi way anova

Recommended readings

- **Getting it right**: R&D Methods for Science and Engineering, Peter Bock, Academic Press
- **The Visual Display of Quantitative Information** Edward R. Tufte

main concepts
experiment statement
factors, modalities
processing
processing steps
parallelization, independance, sequentiality
factorial tree
data / observations
Main steps
definition computation mining reporting
distant computing data retrieval

References

[1] P. Bock and B. Scheibe. *Getting it right: R & D methods for science and engineering*. Academic Press, 2001.

[2] R. D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[3] P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research in signal processing. *Signal Processing Magazine, IEEE*, 26(3):37–47, May 2009.