expLanes: beautiful computational experiments

Mathieu Lagrange

November 30, 2015

About you

Let's assume that you 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 data 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 between several projects is sometimes too long with respect to the time you can allocate to reach efficiency. 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 you are interested in.

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 sharable state[1] ?

[1] `http://sciencecodemanifesto.org`

If so, please consider giving expLanes 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].

[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

expLanes is a software framework currently implemented in Matlab that

- provides a high level abstraction for running computational experiments
- allows multiple users per experiment
- allows multiple processing platforms 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 needed to solve them 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 meaningfulness of the research results that are nowadays published. Admittedly though, following strictly the scientific method can be tedious and shortcuts might be tempting. expLanes is designed to assist you in the most tedious and most 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 expLanes framework can be helpful during this iteration process.

The scientific method[3]

Phases	Steps	expLanes
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

expLanes operates in the Matlab environment. A recent version of this software (> R2014) is needed as well as the Statistical toolbox. The Parallel toolbox is needed for in session multi-core processing.

As much as possible, expLanes uses built in tools to manipulate data. However some functionality are handled through the use basic Unix tools:

1. **pdflatex**: handles compilation of reports written in LaTeX.
2. **rsync**: allow backup, generation of bundles and data communication between hosts.
3. **ssh**: allow secure connection between host.
4. **screen**: allow handling of several detached terminals for background processing.

Please note that those tools must be available on every host where an expLanes project is processed. That said, expLanes do not enforce their availability. For example, without `pdflatex`, no reports can be generated.

Probing access of those tools by your expLanes project (in this documentation, the project we will demonstrate is called `geometricShape`) can be done using the following command:

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

It also check if the different data paths and processing hosts are achievable. Each of those configuration parameters can be set for each user by editing the configuration file of the project. A default configuration file that is used as template when creating a new project is available at ~`/.expLanes/<userName>Config.txt`.

Linux

On Linux hosts, the installation should be straightforward. For example, on Debian, Ubuntu systems, the following packages should be sufficient:

```
  texlive-full rsync openssh-server openssh-client ...
screen
```

Installation may be achieved by typing in a terminal:

```
sudo apt-get install <packagesNames>
```

Minimal configuration might be needed to configure ssh on client and server sides depending on your network configuration. In expLanes ease of use is achieved by using empty passphrase logging.

Mac OS

Several alternatives are available to install Unix packages: Fink, MacPorts and Homebrew. You shall avoid the use of `/usr` for installation, and the `PATH` environment variable shall be properly set.

Windows

For pdflatex, please use MiKTeX, for the remaining please use Cygwin. Some tweaks might be needed in order to ensure that the paths and usernames are correct.

expLanes in a nutshell

expLanes 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. For reference, this project is available in the `demonstrations` directory.

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

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

Create project

Let us call create the project *geometricShape*:

```
expCreate('geometricShape');
```

The command ends by moving into the experiment directory.

This documentation can be loaded at any time using the help command:

```
geometricShape('h');
```

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

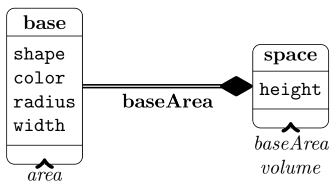
Observations for the 2 processing steps are the following. First step:

```
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);
```

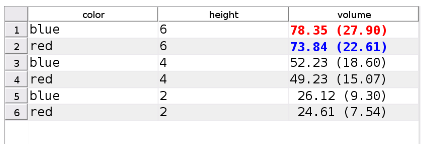
run the base step over all 18 settings.

```
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:



	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)

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. expLanes provides many tools for this purpose, type `help expExpose` for quick reference.

The command:

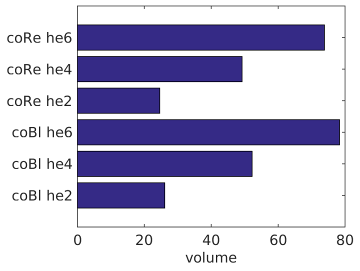
```
geometricShape('display', 2, 'mask', {1 0 1},...
  'expose', {'t', 'obs', 3, 'sort', 1});
```

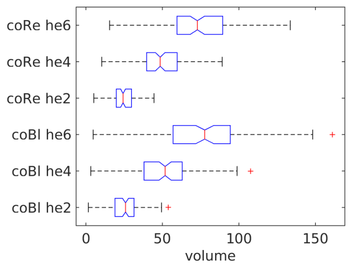
displays the volume (observation 3) of the step volume (2) for each cylinder of radius 2 sorted according to the first on a table (t). Red color indicates best performance, and blue ones, performances which have not been found statistically different from the best one.

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.

Table 1: LaTeXoutput.

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**





Report

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

```
config = expExpose(config, 'l', 'obs', 3, ...
    'mask', {1 0 1}, 'save', 'geol');
config = expExpose(config, 'b', 'obs', 3, ...
    'mask', {1 0 1}, 'save', 'geob', 'orientation', ...
        'h');
config = expExpose(config, 'x', 'obs', 3, ...
    'mask', {1 0 1}, 'save', 'geox', 'orientation', ...
        'h');
```

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. For debug information about the compilation, please add the `'b'` flag to the `report` value.

Demonstrations

Together with examples detailed in this manual, some demonstrations are available in the `demo` directory. At the root of the directory, some reports are given as examples of what can be generated within an expLanes experiment.

Those reports are available online:

1. **Computation of distances:** `https://github.com/mathieulagrange/expLanes/blob/master/demo/distanceComputation.pdf`
2. **Clustering:** `https://github.com/mathieulagrange/expLanes/blob/master/demo/clusteringData.pdf`

3. **Classification:** https://github.com/mathieulagrange/expLanes/blob/master/demo/classifyData.pdf

4. **Audio source separation:** https://github.com/mathieulagrange/expLanes/blob/master/demo/denoiseAudio.pdf

Architecture of an experiment

An expLanes experiment has a specific directory architecture.

Code

The main directory hosts the 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 expLanes experiment has access to a structure named `config` which is imported from a file which is specific to each user. This file is generated at the creation of the experiment from a user specific template file that can be found in ∼/.expLanes .

The syntax of this file allows the user to handle several configurations that depends 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'`, the `codePath` in the config structure is equal to `'~/code/geometricShape'`. If the `codePath` has fewer entries than the one dedicated to the names of the machines, the last entry is chosen. Thus, if the experiment is processed on the machine `'dodo'`, or `'momo'`, the second entry is chosen, that is `'/lab/code/geometricShape'`.

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 for specific user usage:

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

outputs:

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

Data

Access to input data is handled in the configuration file with the entry `inputPath`. Processing data is handled with the entry `dataPath`. Before processing any step, a directory is created with the name of the step and processing data generated by this step is stored here.

Observations are usually of smaller size. As such it may useful to store them in a different location such as a cloud filesystem. The `obsPath` stores the observation data in the same directory architecture. If left empty, the `obsPath` is set equal to `dataPath`.

Report

The report directory stores expositions of observations in many flavors.

- **`report/`**: contains editable LATEXfiles, one for each `reportName` (default to `<projectName>.tex`) and BibTex file that can be edited and processed in a standard fashion. Expositions of observations are added to the LATEXfile prior compilation at the location of the flag `expLanesInsertionFlag`.

- **`report/figures`**: when the `'save'` parameter is added to a visual exposition, the resulting figure is stored in several formats (`.fig` Matlab Figure, `.eps`, `.png`, and `.pdf`). The numerical data is also made available in a `.mat` file.

- **`report/reports`**: contains the generated reports with the following naming : `<reportName><userName><date>.pdf`. The data of every expositions of the corresponding report is made available in a `.mat` file.

- **`report/tables`**: when the `'save'` parameter is added to a table exposition (`'t'` or `'l'`), the resulting table is stored in several formats (LATEXfloating table, LATEXtabulary array, `.csv` file. The numerical data is also made available in a `.mat` file.

- **`report/tex`**: this directory is used for internal expLanes usage. Please do not edit any files in it, as they may be deleted.

Configuration of an experiment

An expLanes experiment can be controlled by editing the configuration file or overloading it with command line settings. Each of those parameters can further be accessed at run time through the variable `config`.

User

- **emailAddress**: email address ticked when processing is done server side (leave empty for no mailing service), can be a cell array of email adresses
- **completeName**: complete name of user

Project

- **probe**: probe tools (1), paths (2), and hosts (3), or all (0)
- **generateRootFile**: regenerate the root file of the experiment (use of newer versions of the expLanes lib may reguired it)
- **addFactor**: add a factor to the experiment. Format of the request: 'name', 'modalities', 'steps', 'selector', defaultModalityId, rankInFactorFile
- **removeFactor**: remove a factor of the experiment. Format of the request: rankInFactorFile, defaultModalityId
- **addStep**: add a step to the experiment. Format of the request: 'name', rank
- **removeStep**: remove a given step of the experiment. Format of the request: rank
- **readMe**: fill the README.txt file with information about replication of the experiment (boolean)

Hostnames and Paths

- **machineNames**: names of the machines as a cell array of 1) string or 2) cell array of strings
- **inputPath**: path to input data (usually accessed read only)
- **codePath**: path to the code repository of the experiment
- **dataPath**: path to the data repository of the experiment (structured with one directory per step)
- **obsPath**: path to the data repository of the experiment for observation data (same as dataPath if left blank)
- **backupPath**: path to backup of transferred data
- **exportPath**: path to export replication of the experiment

- **matlabPath**: path to the directory of the Matlab binary
- **toolPath**: path to Unix tools (pdflatex, rsync, ssh, screen)

Every path can be a cell array of string. In this case, the path is selected in the cell array according to the rank of the host in the machineNames field.

Code

- **dependencies**: code dependencies to load and export as a cell array of strings with paths to the dependencies
- **localDependencies**: dependencies (including expLanes) can be part of the experiment in a dependencies directory: 0 do not use local versions, 1 use local versions, 2 update local versions
- **codeVersion**: version tag of the code of the experiment

Computing

- **do**: processing steps to execute: -1 none, 0 all, >0 processing step by numeric id
- **resume**: do not perform computation if data and obs files with runID>resume are already there
- **parallel**: use parallel processing, if |parallel|>1 specify the number of cores, if array set at step level, if >0 parallelize settings, if <0 parallelize within each setting (the code of the step shall use parfor or the like), can be a numeric value for all processing steps or an array of numeric value, one per step
- **mask**: factor mask: cell array defining the modalities to be set for the factors (0 do all, 1 first modality, [1 3] first and third modality), can be a cell array of cell array
- **design**: use a canonical experimental design: assume cell array of [factors as numericIds], number of values per factor (0 complete set of values), type of plan as string 'f' (factorial) or 'o' (one factor at a time), [seed as a vector of index of factor]
- **dummy**: dummy mode: allow for short computations to dry run the experiment. Set as a numeric value. 0: no dummy mode, >1 generate stored data and observations flagged with the numeric value
- **setRandomSeed**: set the random seed at init for replicability purposes, 0 do not set, >0 set to value.
- **host**: host index to run the experiment: 0: seek by hostName, >0: server mode, <0: local mode, 2.1 means the first host of the second compound (cell array)
- **log**: log level (set to 0 for no log)

- **progress**: show processing progress 0: none, 1: garphical bar if on local mode, 2: verbose output, 3: liminar output
- **exitMatlab**: exit matlab at the end of the computation
- **recordTiming**: show timing observations

Data

- **encodingVersion**: encoding type of mat files (doc save for specs)
- **store**: perform computation to be stored: (1, 0) load data of previous step, (-1, 0) load data result of the current step
- **retrieve**: retrieve needed data server side: -1 no retrieval, 0 global scan of every hosts, >0 hostId
- **namingConventionForFiles**: naming convention for data files: long (complete naming, may lead to too long file names), short (abbreviated naming, may also lead to too long file names), hash (hash based naming, compact but may lead to naming clashes)
- **export**: export a replicate of the experiment with elements designated as a string with tokens separated by white spaces (addition of 'z' outputs a zip file): 'c' (code), 'd' (dependencies), 'i' (input), 1 (output data of step 1), 1d (stored data of step 1 only), 1o (observations of step 1 only)
- **sync**: sync data across machines specified as a cell array with: elements (as in bundle), optional host as numeric id (1.2 means the second host of the first group of machines, default to 2), direction 'd' (download from server to local host, default), 'u' (upload from host to server). Provided with the elements as string, the command default to server 2 in download mode.
- **clean**: clean explanes directories and project data repositories. As string id : t (expLanes temporary directory), b (experiment backup directory), k (all steps directories while keeping reachable settings), 1 (output data of step 1), 1d (remove stored data of step 1 only), 1o (observations of step 1 only). As numeric id (clean the corresponding step directory, 0 means all directories)
- **branchStep**: specify at which step the branching (if any) is effective (default 0)

Exposition

- **display**: step for which to display observations: -1 none, 0 last processed step if any, >0 specific step
- **expose**: specify default display of observations (>: prompt, t: table, p:plot)
- **tableDigitPrecision**: mantissa precision for the display of observations

- **displayFontSize**: font size for the display of observations in figures

Report

- **report**: generate report: combination of r (run report), c (LaTeX compilation), and d (debug output)
- **reportName**: type of report: empty (default), if containing the word <slides> or <Slides>, beamer presentation mode is used
- **latexDocumentClass**: style of LaTeX document (warning: this parameter is taken into account only at the creation of the LaTeX report file)
- **pdfViewer**: path to the pdf viewer (if left empty, expLanes try to locate it automatically)
- **significanceThreshold**: threshold for statistical significance testing
- **showFactorsInReport**: show the factors graph in the report: 0, no display, 1 compact display, 2 alos show stored data, 3 also show observations, 4 also show stored data and observations (if negative only generate the figure for latter inclusion)
- **factorDisplayStyle**: style of the factor graph: 0 no propagation of factors, 1 propagation of factors, 2 propagation of factors with an "all steps" node
- **figureCopyPath**: path where saved figures are copied outside the expLanes project
- **tableCopyPath**: path where saved tables are copied outside the expLanes project

Miscellaneous

- **useExpCodeSmtp**: usage of default expcode smtp. If set to 0 assume availability of local stmp server with complete credentials
- **sendMail**: send email: -2, server mode, email at start and end, -1 server mode, email at end only, 0, no email ever, 1 one email at end, 2 email at start and end

Exposition of observations

To generate Tables, Figures that can be saved or embedded in your reports, the main function to use is:

```
  config = expExpose(config, '<typeOfExposition>', ...
<parameters>);
```

The type of exposition specify the style of rendering. Several predefined styles are available and custom ones can easily be built.

Parameters customizes the rendering and are defined as pair of string / value.

Important: as expLanes do not use global variables, the state of the experiment is entirely stored in the variable `config`. Therefore, this variable has to be propagated trough each of the exposition calls.

Available expositions

Available expositions are:

- **>**: redirect the observations to the command prompt
- **l**: generate a LaTeXtable
- **t**: Matlab table
- **b**: bar plot
- **p**: line plot
- **s**: scatter plot with up to 4 observations respectively mapped to the x-axis, the y-axis, the point size and the color
- **x**: box plot
- **a**: anova display
- **i**: image display

Custom exposition

Let us assume that you need a custom way of displaying observations. You can do so:

```
config = expExpose('custom');
```

At the first call, the user is prompted:

```
Unable to find the function exposeCustom in your path.
This function is needed to display the observations with
the Custom type of display.
Do you want to create it ?  Y/N [Y]:
```

After acknowledgment, the script if generated in the `codePath`. This function has the following signature:

```
config = exposeCustom(config, data, p)
```

where,

- **config**: contains the expLanes configuration state
- **data**: contains the observations as a struct array
- **p**: contains the exposition parameters

Parameters

Parameters customizes the rendering and are defined as pair of string / value.

- **addSpecification**: add display specification to plot directive as ('parameter' / value) pairs; value can be a cell array, in this case the cell array is split across plot items
- **addSettingSpecification**: add display specification to plot; directive relative to specific settings as ('parameter' / value) pairs, value have to be a cell array which is split across plot items which corresponds to the settings
- **caption**: caption of display as string; symbol + gets replaced by a compact description of the setting
- **color**: color of line; 1: default set of colors 0: black 'r', ...: user defined set of colors
- **compactLabels**: shorten labels by removing common substrings (default 0)
- **data**: specify data to be stored (default empty)
- **design**: use a canonical experimental design: assume cell array of [factors as numericIds], number of values per factor (0 complete set of values), type of plan as string 'f' (factorial) or 'o' (one factor at a time), [optionally a seed as a vector of index of factor]. Shortcuts: numerid id (equivalent to [], <numericId>, 'f'), or 'one' design (equivalent to [], 2, 'o'), or 'star' design (equivalent to [], 0, 'o')
- **expand**: name or index of the factor to expand
- **fontSize**: set the font size of LaTeXtables (default 'normal')
- **highlight**: highlight settings that are not significantly different from the best performing setting; -1: no variance, 0: variance for all observations, [1, 3]: variance for the first and third observations
- **highlightStyle**: type of highlighting; 'best': highlight best and equivalents (default), 'Best': highlight only the best, 'better': highlight only the best if significantly better than the others, 'Better': highlight only the best if significantly better than the others and show only this one
- **highlightColor**: use color to show highlights (default 1), -1 do not use * to display the best performing one
- **integrate**: factor(s) to integrate by summation
- **label**: LaTeXlabel of display as string (equal to the name if left empty)

- **legendLocation**: location of the legend (default 'BestOutSide', doc legend for more options)
- **marker**: specification of markers for line plot; 1: Matlab default set of markers (default), 0: no markers, ", ...: user defined cell array of markers
- **mask**: selection of the settings to be displayed
- **mergeDisplay**: concatenate current display with the previous one; ": no merge (default), 'h': horizontal concatenation, 'v': vertical concatenation
- **multipage**: activate the multipage setting to the LATEXtable
- **name**: name of exported file as string; symbol + gets replaced by a compact description of the setting
- **number**: add a line number for each setting in tables
- **noFactor**: remove setting factors
- **noObservation**: remove observations
- **obs**: name(s) or index(es) of the observations to display
- **orderFactor**: numeric array specifying the ordering of the factors
- **orderSetting**: numeric array specifying the ordering of the settings
- **orientation**: orientation of display; 'v': vertical (default), 'h': horizontal, 'i': as second letter invert the table for prompt and latex display
- **percent**: display observations in percent; selector is the same as 'highlight'
- **plotCommand**: set of commands executed after the plotting directives as a cell array of commands
- **plotAxisProperties**: set of command setting the current axis after plotting directives as a cell array of couple property / value (see axis properties in Matlab documentation)
- **precision**: mantissa precision of data as numeric value or array; -1: take value of `config` parameter tableDigitPrecision (default), 0: no mantissa
- **put**: specify display output; 0: otuput to command prompt, 1: output to figure, 2: output to LATEX
- **report**: include the current exposition in the report; 0: do not include (default for Matlab tables), 1: include (default for other displays)
- **rotateAxis**: rotate X axis labels (in degrees)

- **show**: display; 'data': actual observations (default), 'rank': ranking among settings, 'best': select best approaches, 'Best': select the significantly best approach (if any)
- **save**: save the display; 0: no saving, 1: save to a file with the name of the display as filename 'name': save to a file with 'name' as filename
- **shortObservations**: compact observation names
- **shortFactors**: compact factor names
- **showMissingSetting**: show missing settings (default 0)
- **sort**: sort settings acording to the specified set of observations if positive or to the specified set of factors if negative
- **step**: show observations the specified processing step as name or index (default last step)
- **title**: title of display as string, symbol + gets replaced by a compact description of the setting
- **total**: display average or summed values for observations; 'v', 'V': vertical with (v) or without (V) display of settings, 'h', 'H': horizontal with (h) or without (H) display of settings
- **variance**: display variance; selector is the same as 'highlight',
- **visible**: show the figure (default 1 except in save mode)

Publishing your experiment

Publishing your experiment together with your research paper can have a significant impact on the pace at which your research community goes as it allows other researchers to replicate exactly your experiments in an efficient way.

Indeed, any decent size processing chain have a many bells and whistles that needs to be tuned correctly in order to perform as described in your research paper. As it surely took you time and great care, it is very unlikely that another researcher, as skillful as he may be, will dedicate the time needed to do what you did.

As for as the actual code of the experiment is concerned, the experiment can be duplicated and independently executed with the following command:

```
geometricShape('export', 'c d', 'exporPath', ...
    <pathToExportLocation>);
```

Publishing input data

Is the data used in your experiment

- freely available ?
- licensed with creative commons licenses ?

- your property and you have the ability to distribute it freely ?

If yes for any of those items, then your experiment can be replicated entirely which is, by far, the best case. If the data is not yet available online, as of year 2015, a convenient way to publish data online is the Archive repository[4]. Please add the needed documentation on how to retrieve the data online in the `README.txt` file of the experiment.

[4] `https://archive.org`

If the data is of small size, it can be conveniently embedded into the published version of the experiment with the `'i'` token in the export command:

```
geometricShape('export', 'c d i', 'exportPath', ...
    <pathToExportLocation>);
```

Publishing explanes processing data

If you do not wish to publish input data, you can still help others members of the community in quickly gaining information about your work by publishing expLanes processing data.

For example, if the first step of the experiment is to compute features, and you can ensure that the processing is not invertible, you may wish to publish the output of this first step, as it will allow others to replicate the following steps of the experiment:

```
geometricShape('export', 'c d 1d', 'exportPath', ...
    <pathToExportLocation>);
```

The `'1d'` stands for the output data of step 1.

If you do not wish to do that, you can still export the observations of the processing steps in order to provide performance data that is not available in your research paper due to page limit constraints:

```
geometricShape('export', 'c d 1o', 'exportPath', ...
    <pathToExportLocation>);
```

The `'1o'` stands for the observations of step 1.

Ensure replicability

Procedure

Thus, the procedure to safely publish replicable code using expLanes is:

1. Fill replicability informations
   (a) start a Matlab session (if possible with another blank user account)
   (b) type `restoredefaultpath`, in order to remove any implicit dependencies
   (c) run the command you wish other users to execute to replicate your experiment:

   ```
   geometricShape('do', 0, 'report', 'r', 'readMe', ...
       1);
   ```

at termination, the `README.txt` will contain important informations about the soware platform used, such as :

```
------
 Replication informations about the experiment ...
     as of 30-Nov-2015
Command: geometricShape('do', 0, 'report', 'r');
Matlab version: 8.5.0.197613 (R2015a)
Loaded toolboxes:
    - matlab
    - signal_toolbox
    - statistics_toolbox
------
```

2. Create duplicate:
   ```
   geometricShape('export', 'c d i', 'exportPath', ...
       <pathToExportLocation>);
   ```

3. Test duplicate
   (a) start a Matlab session (if possible with another blank user account)
   (b) go the export directory
   (c) optionally, decompress the zip archive
   (d) run the command available in the `README.txt` file.

Advanced usage

Branching experiments

Experimental research most of the time involves the implementation of many solutions that will potentially improve things for a given step of the experiment. Finding the correct way to do it requires a lot of trials that may lead to code burden.

In order to keep a root project clean, and not to replicate the code and data and of this root project in a more experimental project, expLanes allows you to create a branch experiment that accesses the data generated by the root experiment.

Let us assume that we want to create a new way of computing the volume of our geometric shapes. We can do so by creating a branch experiment:

```
   expBranch('branchGeometricShape', <pathToGeometricShape>, ...
2)
```

The last parameter specifies at which step the branching is done, in this case at step 2. The new experiment must keep the same factors up to the step before the branching step (in our case step 1).

The insertion of the new solution into the root experiment must be done manually.

Design of experiments

By default, expLanes proceed to the factorial exploration of every settings. The number of settings being the product of the cardinality of each set of modalities corresponding to the factors defined in the Factor file. For example, given 4 factors, each of 10 modalities, the number of settings to process is 1000, which may be impractical for any kind of large scale data processing.

One way is to sample the dataset, *i.e.* reduce the computation time needed for each setting. This can be done by using the dummy mode in the configuration of the experiment.

Another, more systematic, way is to prune out irrelevant sets of settings. To do so, one has to identify a minimal set of settings to process in order to quickly gain insights, given an set of observations of interest, about:

1. what are the most influential factors ?
2. are there any interaction between factors ?

Let us consider an experiment available in the `demo` directory called `designOfExperiments`. It has 4 factors (`f1, f2, f3, ... f4`), each with a set of modalities equal to 10. There is a unique observation of choice for this experiment (named `metric`) and the relationship between this observation and the factors is the following:

In this example, the answer to the 2 previous questions can easily be deduced. Factor `f4` do not need to be evaluated and factors `f1` and `f2` are correlated and thus have to be jointly considered.

Yet, in most systems, those facts cannot be easily deduced. Considering a reduced set of modalities allows the experiment to be tractable. This can be done by considering the `'design'` parameter. A factorial design with 2 values (minima and maxima) for each modality can be done using the following command

```
designOfExperiments('do', 1, 'design', 2);
```

which is equivalent to

```
designOfExperiments(('do', 1, 'design', {[], 2, 'f'});
```

the empty array means consideration of each setting and 'f' stand for a factorial exploration.

```
designOfExperiments('expose', 'a', 'design', 2);
```

allows the user to study factors relationships through a multiway factorial anova analysis, where the correlation between `f1` and `f2` and the uselessness of `f4` are shown. In order to identify the best setting, the remaining step are:

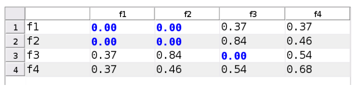
```
designOfExperiments('do', 1, 'mask', {0 0 1 1});
```

The best performance is given for `f1=10` and `f2=10`. Then,

```
designOfExperiments('do', 1, 'mask', {10 10 0});
```

gives the best performance for `f3=10`. Using this method, the optimal setting (10 10 0 *) is identified in $2^4 + 10^2 + 10 = 126$ computation of settings compared to $10^4 = 10000$ for an exhaustive factorial exploration.

Alternative designs can be generated. The "one factor at a time" design allows you to explore the factors considering they are not correlated. In our example



		f1	f2	f3	f4
1	f1	**0.00**	**0.00**	0.37	0.37
2	f2	**0.00**	**0.00**	0.84	0.46
3	f3	0.37	0.84	**0.00**	0.54
4	f4	0.37	0.46	0.54	0.68

```
designOfExperiments('design', 'one', 'do', 1);
```

is equivalent to

```
designOfExperiments('mask', {{[1 10] 1 1 1}, {1 [1 ...
    10] 1 1}, {1 1 [1 10] 1}, {1 1 1 [1 10]}}, 'do', 1);
```

A "star" exploration can be done using

```
designOfExperiments('design', 'star', 'do', 1);
```

which is equivalent to

```
designOfExperiments('mask', {{0 1 1 1}, {1 0 1 1}, ...
    {1 1 0 1}, {1 1 1 0}}, 'do', 1);
```

If available, a initial guess can be provided :

```
designOfExperiments('design', {[], 0, 'o', {5 4 3 ...
    2}}, 'do', 1);
```

where the empty array stands for the exploration of each factor, 0 stands for the complete exploration of the modalities, `'o'` stands for "one factor at a time" exploration, and the last parameter is the (optional) seed. This last command is equivalent to:

```
designOfExperiments('mask', {{0 4 3 2}, {5 0 3 2}, ...
    {5 4 0 2}, {5 4 3 0}}, 'do', 1);
```

The design of experiments method can provide answers for other important questions. Barr & al. wrote a practical introduction on good practices in designing computational experiments [5]. Some books provide more in depth presentation [6].

[5] RichardS. Barr, BruceL. Golden, JamesP. Kelly, MauricioG.C. Resende, and Jr. Stewart, WilliamR. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995

[6] Scott E. Maxwell, Harold D. Delaney, and Ken Kelley. *Designing Experiments and Analyzing Data: A Model Comparison Perspective, Second Edition*. Routledge, 2003; and George E. P. Box, J. Stuart Hunter, and William G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery, 2nd Edition*. Wiley-Interscience, 2005

Handling compilation directives

The expLanes experiment may depends on mex files or external libraries that have to be compiled before the experiment is run.

If the code is static, and each processing host share the same computing environment, the compilation can be done once. If not, you may use the `expMex` function, placed in the Init file of the experiment. Then this function will, before any computation, proceed to the compilation if needed.

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
- **Warning Signs in Experimental Design and Interpretation** Peter Norvig.

`http://norvig.com/experiment-design.html`