CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Civil Engineering
Department of mechanics
Thákurova 7
166 29 Praha 6

Issue No. 1
Revision No. 0
Date 30.01.2024
Number of pages: 11

MixedDoE-MATLAB

Documentation and software manual

MATĚJ LEPŠ
ANNA KUČEROVÁ

Contents

Introduction

MixedDoE-MATLAB is a program that creates a Design of (computer) Experiments (DoE) for combination of discrete and/or continuous variables in Matlab. The discrete part is composed either of full factorial or Taguchi-type Orthogonal array designs. For each point in the discrete part, a sliced design [1] is created for continuous variables. In this way, the continuous domain is uniformly covered but still, for different discrete points the continuous part is not overlapping. Two procedures for the continuous part are available - either Halton sequence, or the classical LHS design. In the same way, the possibility of independent testing data generation is available.

The software is available at `https://github.com/lepsmate/MixedDoE-MATLAB`. It is provided under LGPL-2.1 license.

Acknowledgment

The program needs a code for Least common multiple and regular Orthogonal array codes. We provide both algorithms that have been found on internet. This way, We would like to deeply thank their authors for providing them to the public.

Financial support for this work was provided by the Technology Agency of the Czech Republic (TAČR), project number FW04020163 (Extension of the HiStruct platform by optimisation of global stability and analysis of design combinations).

Chapter 1

Usage

The program only needs a JSON input file that defines the settings of the DoE. Then, the program is run in Matlab environment by typing `>> main 'YourFile.json'`. See the next section for hints how to create an input file, or inspect Examples section 1.3 for an idea, how an input file looks like. Section 1.2 then describes individual files for detailed reference.

1.1 Creating the input file

1.1.1 Preambule

Input file should be in JSON format `https://en.wikipedia.org/wiki/JSON`. Our input starts with following lines:

Lines 1-8 of the `example.json` file

```
1 {
2  "Help": ...,
3  "TypeOfDoEs":"dependent",
4  "TypeOfOA":"fullfact",
5  "TypeOfLHS":"halton",
6  "TypeOfRequest":"testtoo",
7  "NumberOfContinuousPoints":10,
8  "NumberOfContinuousTestPoints":10,
```

- `"Help"` line in our Examples usually contains all possible keywords
- `"TypeOfDoEs"` has two possibilities:
  - `"dependent"` is the default settings that creates combination of all variables
  - `"independent"` is reserved for future use if somebody needs discrete and continuous variables independently. This variant is not implemented yet.
- `"TypeOfOA"`
  - `"fullfact"` is the full factorial design for all discrete variables. The size is a multiple of all levels.

  - `"OA"` is Taguchi style Orthogonal Array for mixed levels. In small dimensions can be same as `"fullfact"`, but in higher dimensions with a lot of levels, it produces dramatically smaller designs.

- `"TypeOfLHS"`
  - `"halton"` generates Halton quasi-random sequence for all continuous variables. Nice property of Haltonn sequence is that every new point "fills-in" the previous ones. In this way, the 'slicing' is produced.
  - `"LHS"` is the classical Latine Hypercube Design from Matlab. It is not appropriate to use it in the context of the mixed DoE, will be redone in the next version.

- `"TypeOfRequest"`
  - `"onlytrain"` Produces just requested design. It will be stored in a `YourFile.out.json` file.
  - `"testtoo"` Produces also testing design, which should be different in continuous variables. It will be found in separate file with a name `YourFile.test.json`

- `"NumberOfContinuousPoints"` defines number of points in the domain of all continuous variables. Be very careful, because the whole design will have the size of **the number of discrete points** *times* **the number of continuous points!**

- `"NumberOfContinuousTestPoints"` has the same meaning as above, but for the case of training set switch on by `"testtoo"` in `"TypeOfRequest"`.

1.1.2 List of variables

The definition of variables follows in `"SetOfInputParameters"` section.

Lines 9-end of the `example.json` file

```
9   "SetOfInputParameters":[
10  ]
```

Each variable has its own section. The order of variables can be arbitrary. The maximum number of variables is limited just by memory of your computer, since the discrete designs are very demanding.

1.1.3 Definition of continuous variables

An example of one continuous variable follows:

Lines 9-end of the example.json file

```
  "SetOfInputParameters":[    {
      "Name":"frame-width",
      "OptimizationType":"continuous",
      "ContinuousMin":6.0,
      "ContinuousMax":60.0
 }
  ]
}
```

Here, "Name" is just for you to easily recognize individual variables, "OptimizationType":"continuous", sets the continuous type and "ContinuousMin" and "ContinuousMax" serve for setting of lower and upper bounds of your variable, respectively.

An example of one discrete variable follows:

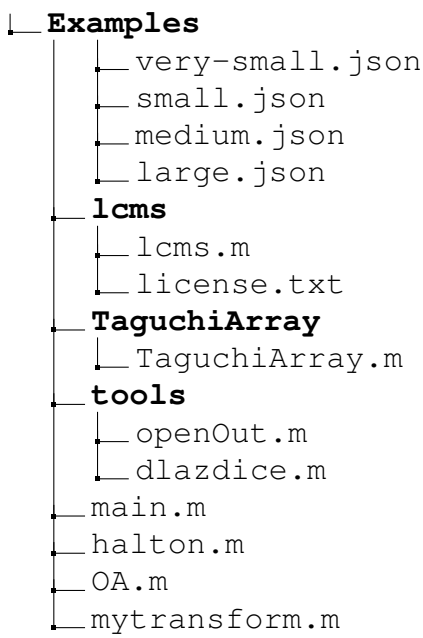
Lines 9-end of the example.json file

```
  "SetOfInputParameters":[    {
      "Name":"roof-slope",
      "OptimizationType":"discrete",
      "DiscreteValuesCount":2
    },
  ]
}
```

Here, "Name" is just for you to easily recognize individual variables, "OptimizationType":"discrete", sets the discrete type and "DiscreteValuesCount" sets the number of levels for your variable. The software just produces numbers 1...DiscreteValuesCount.

1.2 Files

In this paragraph, the files available are briefly described with the following folder tree:



1.2.1 Example files

There are four examples that we have used for testing of the program.

1.2.2 lcms files

script lcms

InputVar: `numberArray` (vector of number of levels).
Code that provides Least common multiple of a given vector. License to this file is provided.

1.2.3 TaguchiArray files

script TaguchiArray

InputVar: `Q` (is the number of the levels), `N` (is the number of the factors (columns)).
Code that provides Taguchi Orthogonal Array for same levels. Reference to the origin of the file is provided in its header.

1.2.4 tools files

Offers two files for working with DoEs:

script openOut

InputVar: `name` (name of the output file with DoE).
Open and read JSON output file. For testing purposes.

script dlazdice

InputVar: `X` (DoE), `group` (OPTIONAL, id for each point in DoE with the group id; each group will have different color; can be used to draw training and testing sets differently, see e.g. Fig. 1.1).
Displays N-dimensional DoE as 2D tiles.

1.2.5 Sampling methodology

script main

The main script of the implemented methodology.
InputVar: `name` (input file name).

The code is driven by the setting of the input file, see Section 1.1.

function X = halton(np,dim)

Generates Halton quasi-random sequence.
InputVar: `np` (number of points), `dim` (dimension).
OutVar: `X` (resulting sequence).

`function TGB = OA(LEVELS)`

Generates TaguchiArray with mixed levels.
InputVar: `LEVELS` (vector of levels for each dimension, e.g. `LEVELS = [2 4 3]`).
OutVar: `TGB` (resulting design).

`function Out = mytransform( DoE, oldmin, oldmax, newmin, newmax )`

Transforms a DoE from old bounds to new ones.
InputVar: `DoE` (Design of Experiments), `oldmin, oldmax, newmin, newmax` (old and new bounds for the DoE, can be scalar as well as vectors).

Examples:
for any size: `mytransform( DoE, 0, 1, -1, 1)`
e.g. for 2D mixed: `mytransform( DoE, 0, 1, [0, 0], [2,5])`
e.g. for 2D vectors: `mytransform( DoE, [-1,-1], [1,1], [0, 0], [2,5])`

1.3 Example

This section shows a usage of proposed methodology demonstrated on a simple example. It is stored in `./Examples/very-small.json` file:

Lines 1-end of the `./Examples/very-small.json` file

```
{
  "Help":"Type of DoEs: ... ,
  "TypeOfDoEs":"dependent",
  "TypeOfOA":"fullfact",
  "TypeOfLHS":"halton",
  "TypeOfRequest":"testtoo",
  "NumberOfContinuousPoints":10,
  "NumberOfContinuousTestPoints":10,
  "SetOfInputParameters":[
    {
      "Name":"frame-width",
      "OptimizationType":"continuous",
      "ContinuousMin":6.0,
      "ContinuousMax":60.0
    },
    {
      "Name":"frame-height",
      "OptimizationType":"continuous",
      "ContinuousMin":3.0,
      "ContinuousMax":25.0
    },
    {
      "Name":"roof-slope",
      "OptimizationType":"discrete",
      "DiscreteValuesCount":1
    },
    {
      "Name":"bay-width",
      "OptimizationType":"continuous",
      "ContinuousMin":2.0,
      "ContinuousMax":12.0
    },
    {
      "Name":"snow-load",
      "OptimizationType":"discrete",
      "DiscreteValuesCount":1
    },
    {
      "Name":"wind-load",
      "OptimizationType":"discrete",
      "DiscreteValuesCount":1
    },
    {
      "Name":"live-load",
      "OptimizationType":"discrete",
      "DiscreteValuesCount":2
    }
  ]
}
```

It contains three continuous variables with different bounds and four discrete variables. The example is aimed to show the space-filling property of the proposed mixed design. In the discrete space, there are just two points, therefore, the code produces four different continuous designs: two for two points in the training set, two for two points in the testing set. The resulting design should look like Fig. 1.1.

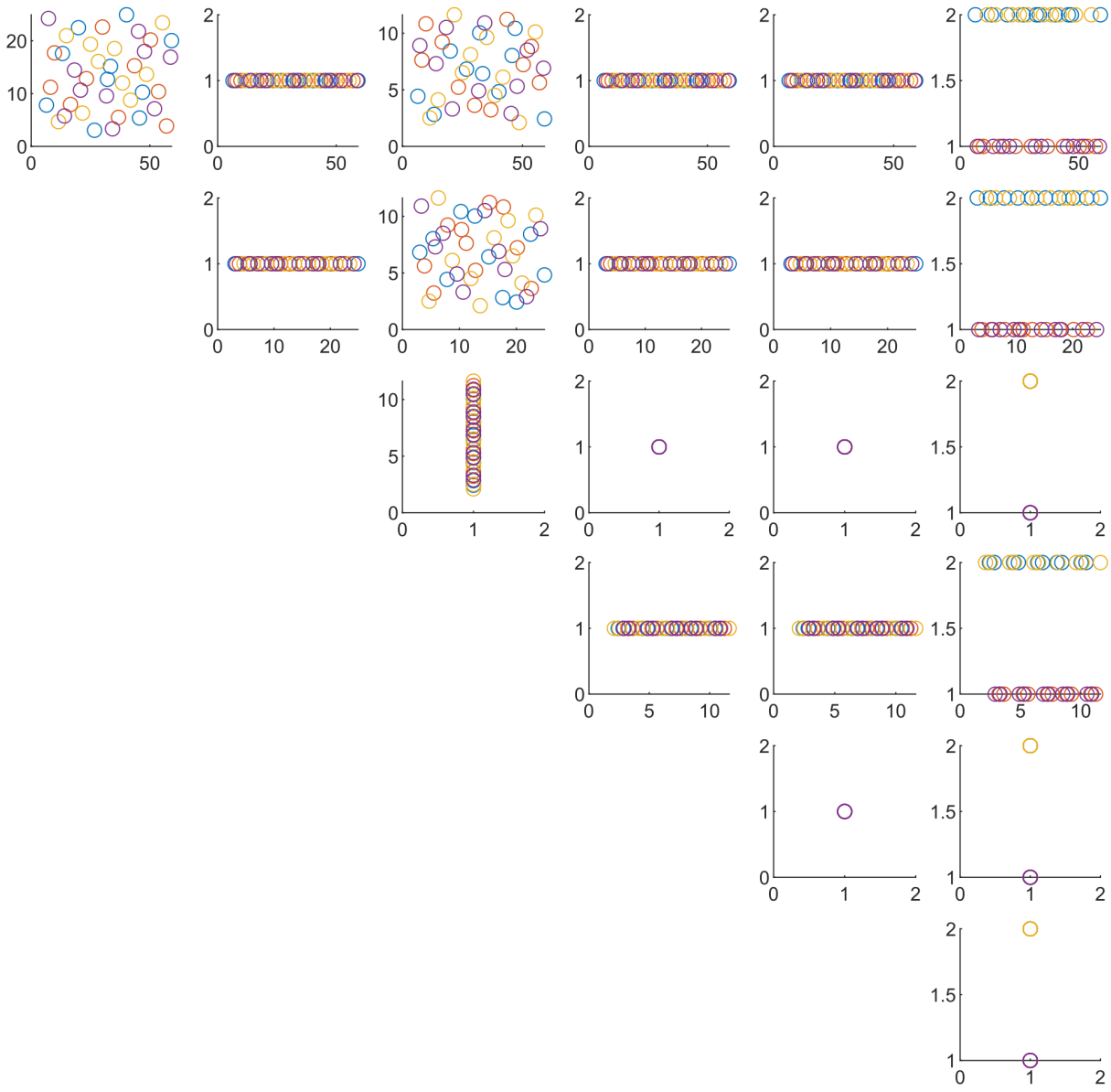


Figure 1.1: Resulting Design of Experiments.

Bibliography

[1] Peter Z. G. Qian. Sliced latin hypercube designs. *Journal of the American Statistical Association*, 107(497):393–399, 2012. DOI: https://doi.org/10.1080/01621459.2011.644132.