doepy

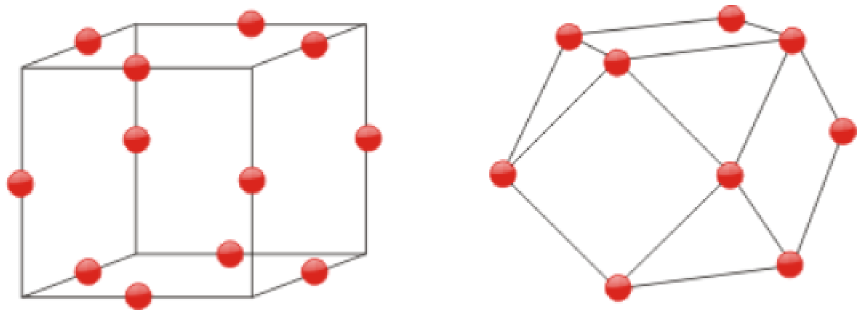
Release 0.0.1

Jul 25, 2019

Contents

CHAPTER 1

Design of Experiment Generator in Python (`pip install doepy`)



Authored and maintained by Dr. Tirthajyoti Sarkar, Fremont, California.

Check my website for detailes about my other projects and data science/ML articles.

CHAPTER 2

Introduction

Design of Experiment (DOE) is an important activity for any scientist, engineer, or statistician planning to conduct experimental analysis. This exercise has become **critical in this age of rapidly expanding field of data science and associated statistical modeling and machine learning**. A well-planned DOE can give a researcher meaningful data set to act upon with optimal number of experiments preserving critical resources.

> After all, aim of Data Science is essentially to conduct highest quality scientific investigation and modeling with real world data. And to do good science with data, one needs to collect it through carefully thought-out experiment to cover all corner cases and reduce any possible bias.

CHAPTER 3

How to use it?

3.1 What supporitng packages are required?

First make sure you have all the necessary packages installed. You can simply run the .bash (Unix/Linux) and .bat (Windows) files provided in the repo, to install those packages from your command line interface. They contain the following commands,

```
pip install numpy
pip install pandas
pip install pydoe
pip install diversipy
```

3.2 How to install the package?

You can pip install the package!

`pip install doepy`

3.3 Github

The package is hosted at this Github repo.

3.4 Quick start

Let's say you have a design problem with the following table for the parameters range. Imagine this as a generic example of a checmical process in a manufacturing plant. You have 3 levels of `Pressure`, 3 levels of `Temperature`, 2 levels of `FlowRate`, and 2 levels of `Time`.

Pressure	Temperature	FlowRate	Time
40	290	0.2	5
50	320	0.3	8
70	350	•	•

First, import `build` module from the package,

```
from doepy import build
```

Then, try a simple example by building a **full factorial design**. We will use `build.full_fact()` function for this. You have to pass a dictionary object to the function which encodes your experimental data.

```
build.full_fact(
{'Pressure':[40,55,70],
'Temperature':[290, 320, 350],
'Flow rate':[0.2,0.4],
'Time':[5,8]}
)
```

If you build a full-factorial DOE out of this, you should get a table with 3x3x2x2 = 36 entries.

Pressure	Temperature	FlowRate	Time
40	290	0.2	5
50	290	0.2	5
70	290	0.2	5
40	320	0.2	5
…	…	…	…
…	…	…	…
40	290	0.3	8
50	290	0.3	8
70	290	0.3	8
40	320	0.3	8
…	…	…	…
…	…	…	…

There are, of course, half-factorial designs to try!

3.5 Latin Hypercube design

Sometimes, a set of **randomized design points within a given range** could be attractive for the experimenter to asses the impact of the process variables on the output. Monte Carlo simulations are close example of this approach.

However, a Latin Hypercube design is better choice for experimental design rather than building a complete random matrix as it tries to subdivide the sample space in smaller cells and choose only one element out of each subcell. This way, a more **uniform spreading' of the random sample points** can be obtained.

User can choose the density of sample points. For example, if we choose to generate a Latin Hypercube of 12 experiments from the same input files, that could look like,

```
build.space_filling_lhs(
{'Pressure':[40,55,70],
```

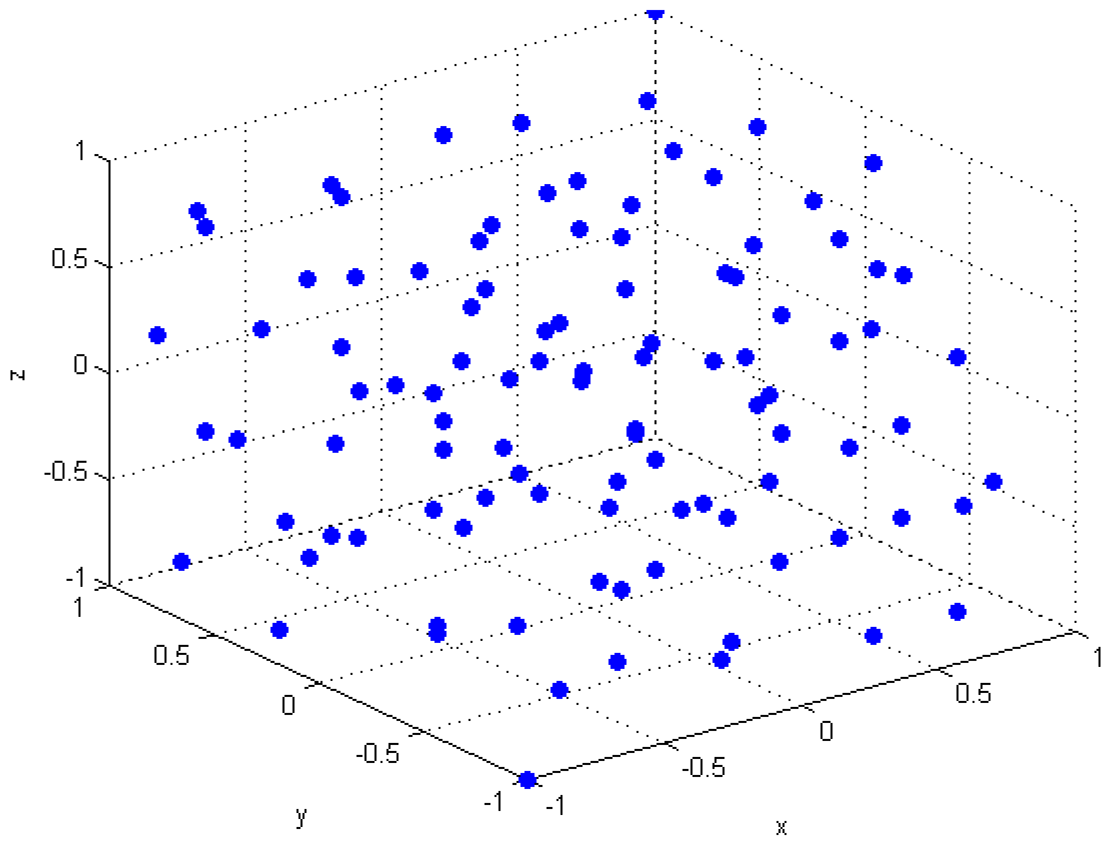
(continues on next page)

(continued from previous page)

```
'Temperature':[290, 320, 350],
'Flow rate':[0.2,0.4],
'Time':[5,11]},
num_samples = 12
)
```

Pressure	Temperature	FlowRate	Time
63.16	313.32	0.37	10.52
61.16	343.88	0.23	5.04
57.83	327.46	0.35	9.47
68.61	309.81	0.35	8.39
66.01	301.29	0.22	6.34
45.76	347.97	0.27	6.94
40.48	320.72	0.29	9.68
51.46	293.35	0.20	7.11
43.63	334.92	0.30	7.66
47.87	339.68	0.26	8.59
55.28	317.68	0.39	5.61
53.99	297.07	0.32	10.43

Of course, there is no guarantee that you will get the same matrix if you run this function because this are randomly sampled, but you get the idea!



3.6 Other functions to try on

Try any one of the following designs,

- Full factorial: `build.full_fact()`
- 2-level fractional factorial: `build.frac_fact_res()`
- Plackett-Burman: `build.plackett_burman()`
- Sukharev grid: `build.sukharev()`
- Box-Behnken: `build.box_behnken()`
- Box-Wilson (Central-composite) with center-faced option: `build.central_composite()` with `face='ccf'` option
- Box-Wilson (Central-composite) with center-inscribed option: `build.central_composite()` with `face='cci'` option
- Box-Wilson (Central-composite) with center-circumscribed option: `build.central_composite()` with `face='ccc'` option
- Latin hypercube (simple): `build.lhs()`
- Latin hypercube (space-filling): `build.space_filling_lhs()`
- Random k-means cluster: `build.random_k_means()`
- Maximin reconstruction: `build.maximin()`
- Halton sequence based: `build.halton()`
- Uniform random matrix: `build.uniform_random()`

3.7 Read from and write to CSV files

Internally, you pass on a dictionary object and get back a Pandas DataFrame. But, for reading from and writing to CSV files, you have to use the `read_write` module of the package.

```
from doepy import read_write
data_in=read_write.read_variables_csv('../Data/params.csv')
```

Then you can use this `data_in` object in the DOE generating functions.

For writing back to a CSV,

```
df_lhs=build.space_filling_lhs(data_in,num_samples=100)
filename = 'lhs'
read_write.write_csv(df_lhs,filename=filename)
```

You should see a `lhs.csv` file in your directory.

3.8 A simple pipeline for building a DOE table

Clubbing together the `build` functions and the `read_write` module, one can devise a simple pipeline to build a DOE from a CSV file input.

Suppose, you have a file called *ranges.csv*, which contains min/max values of an arbitrary number of parameters, in your directory. Just two lines of code will generate a space-filling Latin hypercube design based on this file with 100 randomized samples spanning over the min/max ranges.

```
from doepy import build, read_write

read_write.write_csv(
build.space_filling_lhs(read_write.read_variables_csv('ranges.csv'),
num_samples=100),
filename='DOE_table.csv'
)
```

CHAPTER 4

Features

At its heart, `doepy` is just a collection of functions, which wrap around the core packages (mentioned below) and generate **design-of-experiment (DOE) matrices** for a statistician or engineer from an arbitrary range of input variables.

4.1 Limitation of the foundation packages used

Both the core packages, which act as foundations to this repo, are not complete in the sense that they do not cover all the necessary functions to generate DOE table that a design engineer may need while planning an experiment. Also, they offer only low-level APIs in the sense that the standard output from them are normalized numpy arrays. It was felt that users, who may not be comfortable in dealing with Python objects directly, should be able to take advantage of their functionalities through a simplified user interface.

4.2 Simplified user interface

There are other DOE generators out there. But they generate n-dimensional arrays. `doepy` is built on the simple theme of being intuitive and easy to work with - for researchers, engineers, and social scientists of all background - not just the ones who can code.

User just needs to provide a simple CSV file with a single table of variables and their ranges (2-level i.e. min/max or 3-level).

Some of the functions work with 2-level min/max range while some others need 3-level ranges from the user (low-mid-high). Intelligence is built into the code to handle the case if the range input is not appropriate and to generate levels by simple linear interpolation from the given input.

The code will generate the DOE as per user's choice and write the matrix in a CSV file on to the disk.

In this way, **the only API user needs to be exposed to, are input and output CSV files**. These files then can be used in any engineering simulator, software, process-control module, or fed into process equipments.

4.3 Pandas DataFrame support

Under the hood, `doepy` generates Numpy arrays and convert them to Pandas DataFrame. Therefore, programatically, it is simple to get those Numpy arrays or DataFrames to do more, if the user wishes so.

4.4 Coming in a future release - support for more types of files

Support for more input/output types will come in future releases - MS Excel, JSON, etc.

4.5 Designs available

- Full factorial,
- 2-level fractional factorial,
- Plackett-Burman,
- Sukharev grid,
- Box-Behnken,
- Box-Wilson (Central-composite) with center-faced option,
- Box-Wilson (Central-composite) with center-inscribed option,
- Box-Wilson (Central-composite) with center-circumscribed option,
- Latin hypercube (simple),
- Latin hypercube (space-filling),
- Random k-means cluster,
- Maximin reconstruction,
- Halton sequence based,
- Uniform random matrix

CHAPTER 5

About Design of Experiment

5.1 What is a scientific experiment?

In its simplest form, a scientific experiment aims at predicting the outcome by introducing a change of the preconditions, which is represented by one or more independent variables, also referred to as “input variables” or “predictor variables.” The change in one or more independent variables is generally hypothesized to result in a change in one or more dependent variables, also referred to as “output variables” or “response variables.” The experimental design may also identify control variables that must be held constant to prevent external factors from affecting the results.

5.2 What is Experimental Design?

Experimental design involves not only the selection of suitable independent, dependent, and control variables, but planning the delivery of the experiment under statistically optimal conditions given the constraints of available resources. There are multiple approaches for determining the set of design points (unique combinations of the settings of the independent variables) to be used in the experiment.

Main concerns in experimental design include the establishment of validity, reliability, and replicability. For example, these concerns can be partially addressed by carefully choosing the independent variable, reducing the risk of measurement error, and ensuring that the documentation of the method is sufficiently detailed. Related concerns include achieving appropriate levels of statistical power and sensitivity.

Need for careful design of experiment arises in all fields of serious scientific, technological, and even social science investigation—*computer science, physics, geology, political science, electrical engineering, psychology, business marketing analysis, financial analytics*, etc…

5.3 Options for open-source DOE builder package in Python?

Unfortunately, majority of the state-of-the-art DOE generators are part of commercial statistical software packages like JMP (SAS) or Minitab. However, a researcher will surely be benefited if there exists an open-source code which presents an intuitive user interface for generating an experimental design plan from a simple list of input variables.

There are a couple of DOE builder Python packages but individually they don't cover all the necessary DOE methods and they lack a simplified user API, where one can just input a CSV file of input variables' range and get back the DOE matrix in another CSV file.

CHAPTER 6

Acknowledgements and Requirements

The code was written in Python 3.7. It uses following external packages that needs to be installed on your system to use it,

- `pydoe`: A package designed to help the scientist, engineer, statistician, etc., to construct appropriate experimental designs. Check the docs here.
- `diversipy`: A collection of algorithms for sampling in hypercubes, selecting diverse subsets, and measuring diversity. Check the docs here.
- `numpy`
- `pandas`