DOE Toolkit© User Guide

Version 3.0

Table of Contents

The code for all the examples presented in this document is included in the companion ***DOE Toolkit User Guide 3.0.nb*** Mathematica® notebook.

Introduction

DOE Toolkit provides six functions that allow you to generate experimental designs for the following design methodologies:

Screening: **DOEFactorial**, **DOEFractional**

Response Surface: **DOECentralComposite**, **DOEBoxBehnken**

Mixture: **DOEMixture**

Orthogonal: **DOEOrthogonal**

Each DOE Toolkit function provides options that allow you to add blocking to any design and to randomize the runs. With the exception of the **DOEMixture** function, each DOE Toolkit function provides options that allow you to add central points and axial points to your design.

A DOE Toolkit function returns the experimental design that it creates in the form of an *XDesign* object. An *XDesign* object is simply a 2-part list; the first part is a Dataset object and the second part is a list of metadata that describes the generated design.

You can also augment a design that was created by one of the DOE Toolkit functions by using the **DOEAugment** function. You supply the XDesign object that was returned by the DOE Toolkit function you used to create the design as input to the **DOEAugment** function. See the Augmenting a Design section for more information.

Once you have created a design using one of the DOE Toolkit functions, you can use the **DOEOptimal** function to find an optimal subset of runs based on that design.

After you have performed the experiment based on your design and recorded the experimental results, the **DOEDesirability** function allows you to visually explore the consequences of changing factor settings in your model.

If you need to edit the Dataset object component of an XDesign object (for example, to add extra rows or to modify an existing row or to add a new column), the DOE Toolkit provides a set of examples that illustrate how to perform the most common row and column operations with a Dataset. For more information, see Appendix 3: Editing a Dataset.

The DOE Toolkit v3.0 runs on Windows, Macintosh, and Linux and is compatible with Mathematica 12.0 or higher.

Please note: In most of the examples in this document as well as in the companion ***DOE Toolkit User Guide 3.0.nb*** notebook, we set the `RandomizeRuns` option (which is an option shared in common by all DOE Toolkit functions) to False. This is done for the purpose of making the table that results from evaluating a given DOE Toolkit function easier to inspect and ensures that the same table results if the same function is repeatedly evaluated. As a general rule, when you use the DOE Toolkit functions in your work, the `RandomizeRuns` option should be set to **True** (which is its default value). The easiest way to ensure that this is the case is to simply avoid specifying the `RandomizeRuns` option.

DOE Toolkit Setup

There is no installer for the DOE Toolkit. Instead, you download the DOEToolkit.zip file and unzip the zip file.

You can obtain the DOEToolkit.zip file from the Demo page at the Harper Corditt Software website.

You can also obtain the DOEToolkit.zip file when you purchase the product at the Wolfram Research website. If you purchase the product from Wolfram Research, you will be provided with a link to download the DOEToolkit.zip file. However, there is no difference between the DOEToolkit.zip file that you obtain from Wolfram Research vs. the DOEToolkit.zip file that you can download from the Demo page at the Harper Corditt Software website.

After you unzip the zip file, a folder named **DOEToolkit** will be created. You should copy the **DOEToolkit** folder to your hard drive. The preferred location on your hard drive is the Wolfram Applications directory.

You can determine the location of the Wolfram Applications directory by evaluating the following line in your notebook:

```
FileNameJoin[{$UserBaseDirectory, "Applications"}]
```

For example, on a Macintosh, evaluating the line above produces output that looks like this:

"/Users/yourNameHere/Library/Wolfram/Applications"

The **DOEToolkit** folder contains of the following files:

DOEToolkit.wl
DOE Toolkit User Guide 3.0.nb
DOE Toolkit User Guide 3.0.pdf

A brief description of these files is given below:

DOEToolkit.wl is a Wolfram Language Package (**.wl** file extension) that contains the definitions of the functions provided by the DOE Toolkit.

DOE Toolkit User Guide 3.0.nb is a Mathematica notebook that contains working code for all the examples contained in this User Guide.

DOE Toolkit User Guide 3.0.pdf is this document.

The DOE Toolkit that you download will run in *demo mode* which means that it has certain restrictions imposed on the complexity of the designs that can be generated.

To make your copy of the DOE Toolkit fully functional, you must *purchase* the product either at the Wolfram Research website or at the Shop page at the Harper Corditt Software website. After you purchase the product, you must register your purchase on the Register page at the Harper Corditt Software website. Once your purchase and registration have been processed, Harper Corditt Software will email to you a key file named **key.doetk**. You should copy the **key.doetk** key file to the **DOEToolkit** folder in the Wolfram Applications directory. From that point on, whenever the DOE Toolkit is loaded into a notebook, it will be fully functional.

If you do not wish to copy the key file to the Wolfram Applications directory, the DOE Toolkit will run in fully functional mode provided the **key.doetk** key file is located in the same directory as your notebook.

There is no support for locating the key file in any other directory. If you store the **key.doetk** key file in a folder named "XYZ" and then load the DOE Toolkit from a notebook that is not located in the XYZ folder, the DOE Toolkit will only run in demo mode under such a scenario.

When you register your purchase, you must provide your Mathematica LicenseID, which you can obtain by evaluating the following line in any notebook:

```
$LicenseID
```

Loading the DOE Toolkit in Your Notebook

In order to gain access to the definitions provided by the DOE Toolkit, you must load the DOE Toolkit into your notebook.

Suppose you have copied the **DOEToolkit** folder to the Wolfram Applications directory on your computer. In that case, you can use the **Needs** function to load the DOE Toolkit by evaluating the following line:

```
Needs["DOEToolkit`"]
```

In some cases, it may be preferable to copy the **DOEToolkit** folder to some location other than the Wolfram Applications directory. In that case, assuming your current notebook is located in the same directory as the **DOEToolkit.wl** file, you can use the **Get** function to load the DOE Toolkit by evaluating the following line:

```
Get@FileNameJoin[{NotebookDirectory[], "DOEToolkit.wl"}]
```

Loading the DOE Toolkit more than once is not harmful, but it will result in several warning messages. To avoid unintentionally loading the DOE Toolkit multiple times, you can first check whether the expression *DOEFactorial* has been defined. Since the **DOEFactorial** function is exported by the **DOEToolkit.wl** package, this is a reasonable way (although hardly foolproof) to determine whether the DOE Toolkit is already loaded. For example:

```
If[!ValueQ[DOEFactorial, Method -> "SymbolDefinitionsPresent"],
  Get@FileNameJoin[{NotebookDirectory[], "DOEToolkit.wl"}]]
```

Notation Used in This Users Guide

Throughout this document, we use a special font to represent Mathematica expressions that you would enter as input in your notebook. Such code examples are usually also surrounded with a thin blue rectangle like so:

```
fullfactxdsn = DOEFactorial[<|"X1" -> {0.7, 1.9}, "X2" -> {40, 60}, "X3" -> {1.8, 2.3}|>,
RandomizeRuns -> False];
fullfactxdsn[[1]]
fullfactxdsn[[2]] // TableForm
```

Output to your notebook is represented in this document in different ways. If the output is a Mathematica dataset, then we use a screen capture of the Dataset object itself. For example, after evaluating the expression above, the following Dataset object is output to your notebook:

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.9	40	1.8
3	121	0.7	60	1.8
4	221	1.9	60	1.8
5	112	0.7	40	2.3
6	212	1.9	40	2.3
7	122	0.7	60	2.3
8	222	1.9	60	2.3

Other types of non-Dataset output are usually represented by screen captures. The output is surrounded by a thin, red rectangle with rounded corners.

For example, suppose you evaluate the expression below:

```
fullfactxdsn[[2]] // TableForm
```

In your notebook, you should see the following lines:

```
Out[4]//TableForm=
        Design Type → Full Factorial
        Data Representation → DataAreLevelListValues
        Pattern Representation → PatternsAreLevelListIndices
        RandomizeRuns → False
        nRuns → 8
        FFDesignIndices → {{1, 1, 1}, {2, 1, 1}, {1, 2, 1}, {2, 2, 1}, {1, 1, 2}, {2, 1, 2}, {1, 2, 2}, {2, 2, 2}}
        nFactors → 3
        n2level Factors → 3
        LevelLists → {{0.7, 1.9}, {40, 60}, {1.8, 2.3}}
        Columns → {{Name → Pattern}, {Name → X1, Factor → True}, {Name → X2, Factor → True}, {Name → X3, Factor → True}}
```

In our notation, the output in this case would be represented as follows:

```
Out[4]//TableForm=
        Design Type → Full Factorial
        Data Representation → DataAreLevelListValues
        Pattern Representation → PatternsAreLevelListIndices
        RandomizeRuns → False
        nRuns → 8
        FFDesignIndices → {{1, 1, 1}, {2, 1, 1}, {1, 2, 1}, {2, 2, 1}, {1, 1, 2}, {2, 1, 2}, {1, 2, 2}, {2, 2, 2}}
        nFactors → 3
        n2level Factors → 3
        LevelLists → {{0.7, 1.9}, {40, 60}, {1.8, 2.3}}
        Columns → {{Name → Pattern}, {Name → X1, Factor → True}, {Name → X2, Factor → True}, {Name → X3, Factor → True}}
```

As the example below illustrates, we will often highlight part of an expression using the color cyan in order to draw attention to some aspect of the syntax. Although you can color your Mathematica code if you wish, it certainly isn't necessary!

```
fullfactxdsn = DOEFactorial[<|"X1" -> {0.7, 1.9}, "X2" -> {40, 60}, "X3" -> {1.8, 2.3}|>,
RandomizeRuns -> False];
```

When describing the syntax of a rule, we sometimes use "**LHS**" to abbreviate the phrase: "left-hand side" and "**RHS**" to abbreviate the phrase: "right-hand side". For example, in our description of the **`CentralPoints`** option, we say:

> You assign a list of *insertion rules* of the form LHS -> RHS to the **`CentralPoints`** option. The LHS of the rule is either one of the keywords:
> **`{DesignInsertAtTop,DesignInsertAtBottom,DesignInsertInMiddle}`**
> or else a row number, which indicates the insertion location. The RHS of the rule indicates the number of central points to be inserted at the designated location.

Last but not least, you may have noticed that when we present examples of input that involve the arrow symbol, we use the two-character combination "->" rather than a single glyph, such as "→". This makes it possible to copy input examples from this document and paste them directly into Mathematica where they can be evaluated without any modification. Note that if we were to use an

arrow glyph such as "→" to illustrate an input example, although our text might look nicer, it would fail to evaluate properly when pasted into a Mathematica notebook.

For example, you can copy/paste the text below into your Mathematica notebook and it will evaluate without a problem:

```
fullfactxdsn = DOEFactorial[<|"X1" -> {0.7, 1.9}, "X2" -> {40, 60}, "X3" -> {1.8, 2.3}|>,
RandomizeRuns->False]
```

However, if you copy/paste the text below into your Mathematica notebook, it will fail to evaluate. To make matters worse, Mathematica does not alert you to the true cause of the problem and will simply echo your input. Needless to say, this can be quite mystifying:

```
fullfactxdsn = DOEFactorial[<|"X1" → {0.7, 1.9}, "X2" → {40, 60}, "X3" → {1.8, 2.3}|>,
RandomizeRuns → False]
```

Datasets and XDesigns

The functions in the DOE Toolkit return what we call an **XDesign** (E**x**perimental Design) object). An XDesign object is a 2-part list whose first part is a Mathematica **Dataset** object and whose second part is a list of metadata. The Dataset component encapsulates the list of runs that were generated by the DOE Toolkit function. The metadata component describes various properties of the experimental design that was generated by the function.

The **Dataset** function is a very powerful function that was introduced in Mathematica 12.0 and which has been significantly improved in Mathematica 12.1. It would go far beyond the scope of this document to attempt to describe all its features. For more information, please consult the Mathematica documentation: https://reference.wolfram.com/language/ref/Dataset.html

By encapsulating the runs produced by a DOE Toolkit function in the form of Mathematica Dataset, we enable the user to take full advantage of the Dataset object for the purposes of viewing, manipulating, and analyzing their experimental results.

The easiest way to view the XDesign object returned by a DOE Toolkit function is to display its two parts separately. Mathematically automatically formats the first part (the Dataset object) for you. One way to display the second part (the metadata list) is to use the **TableForm** function.

For example, suppose you evaluate the **DOEFactorial** function as shown below to create a full factorial design for 3 variables, each of which has 2 levels:

```
fullfactxdsn = DOEFactorial[<|"X1" -> {0.7, 1.9}, "X2" -> {40, 60}, "X3" -> {1.8, 2.3}|>,
RandomizeRuns -> False];
```

The following lines will display the 2 parts of the XDesign object returned by the **DOEFactorial** function:

```
fullfactxdsn[[1]]
fullfactxdsn[[2]] // TableForm
```

fullfactxdsn[[1]] outputs the Dataset object:

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.9	40	1.8
3	121	0.7	60	1.8
4	221	1.9	60	1.8
5	112	0.7	40	2.3
6	212	1.9	40	2.3
7	122	0.7	60	2.3
8	222	1.9	60	2.3

fullfactxdsn[[2]] // **TableForm** outputs the metadata list in **TableForm**:

```
Out[4]//TableForm=
        Design Type → Full Factorial
        Data Representation → DataAreLevelListValues
        Pattern Representation → PatternsAreLevelListIndices
        RandomizeRuns → False
        nRuns → 8
        FFDesignIndices → {{1, 1, 1}, {2, 1, 1}, {1, 2, 1}, {2, 2, 1}, {1, 1, 2}, {2, 1, 2}, {1, 2, 2}, {2, 2, 2}}
        nFactors → 3
        n2level Factors → 3
        LevelLists → {{0.7, 1.9}, {40, 60}, {1.8, 2.3}}
        Columns → {{Name → Pattern}, {Name → X1, Factor → True}, {Name → X2, Factor → True}, {Name → X3, Factor → True}}
```

If you need to augment your design, the DOE Toolkit provides the **DOEAugment** function to help you make the necessary modifications. For more information, see the Augmenting a Design section.

If you need to edit a Dataset object, for example, to add extra rows or to modify an existing row or to insert a new column, the DOE Toolkit provides a set of examples illustrating how to perform the most common row and column Dataset operations. For more information, see Appendix 3: Editing a Dataset.

Common Features of DOE Toolkit Functions

The following DOE Toolkit functions share several features in common:

DOEBoxBehnken
DOECentralComposite
DOEFactorial
DOEFractional
DOEMixture
DOEOptimal
DOEOrthogonal

With the exception of the **DOEMixture** function, whenever you evaluate a DOE Toolkit function in the list above, the first parameter you supply to the function is always an *association*. Each element of the association is a *rule* of the form: FactorName->ListOfFactorLevels where ListOfFactorLevels is a list of the values that can be achieved by the factor named FactorName. We call the list of values that can be achieved by a given factor its ”level list”.

Although it is not absolutely required, we strongly recommend that you provide each FactorName in the form of a double-quoted string. If you don’t double-quote a factor name and instead use what Mathematica calls a symbol, then it is possible for confusion to result if you re-use that symbol for some other purpose in some other section of your notebook. So, rather than form a rule as: X1->{0.7,1.9}, we recommend that you provide the rule like so: "X1"->{0.7,1.9}.

For example, suppose you evaluate the **DOEFactorial** function as follows:

```
factorialxdsn = DOEFactorial[<|"X1"->{0.7,1.9},"X2"->{40,50,60}, "X3"->{1.8,2.3}|>,
RandomizeRuns->False];
factorialxdsn[[1]]
```

This creates the output shown below. Note: We have also set the **`RandomizeRuns`** option to False, to make it easier to verify that a full factorial design has been produced. Ordinarily, you will want to set the **`RandomizeRuns`** option to True (which is the default).

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.9	40	1.8
3	121	0.7	50	1.8
4	221	1.9	50	1.8
5	131	0.7	60	1.8
6	231	1.9	60	1.8
7	112	0.7	40	2.3
8	212	1.9	40	2.3
9	122	0.7	50	2.3
10	222	1.9	50	2.3
11	132	0.7	60	2.3
12	232	1.9	60	2.3

The **DOEFactorial** function has generated a Dataset object that represents a full factorial design with the following properties:

(1) There are 3 columns for the factors involved, named “X1”, ”X2”, and “X3”.
(2) The “X1” factor is a 2-level factor whose values can be: 0.7 or 1.9.
(3) The “X2” factor is a 3-level factor whose values can be: 40, 50, or 60.
(4) The “X3” factor is a 2-level factor whose values can be: 1.8 or 2.3.

The value returned by a DOE Toolkit function is always an XDesign object. (See the Datasets and XDesigns section for more information about XDesigns.)

Ordinarily, the Dataset object will contain a column on the left which shows row numbers for the runs in the design. You can turn off the column of row numbers by setting the **`RowNumbers`** option to False.

By default, any DOE Toolkit function will represent the data in the resulting Dataset in the form of the actual factor level values involved. You can control this behavior by means of the **`DataRepresentation`** option. For example, you can set this option to **`DataAreLevelListIndices`**, and in that case, the resulting design will display level list indices for the data as shown:

```
factorialxdsn=DOEFactorial[<|"X1"->{0.7,1.9}, "X2"->{40,50,60}, "X3"->{1.8,2.3}|>,
RandomizeRuns->False,
DataRepresentation->DataAreLevelListIndices];
factorialxdsn[[1]]
```

	Pattern	X1	X2	X3
1	111	1	1	1
2	211	2	1	1
3	121	1	2	1
4	221	2	2	1
5	131	1	3	1
6	231	2	3	1
7	112	1	1	2
8	212	2	1	2
9	122	1	2	2
10	222	2	2	2
11	132	1	3	2
12	232	2	3	2

For example, row 11 indicates that the X1 factor is at level 1, the X2 factor is at level 3, and the X3 factor is at level 2. If you prefer to use this form of data representation, then you will probably want to turn off the Pattern column (set the **`PatternColumn`** option to False) since it duplicates the data in the factor columns.

You can also set the **`DataRepresentation`** option to a customized list **H**, whose elements are lists of symbols (aka *symbol-lists*). For any number N, if the design contains a factor that has N levels, then **H** must contain a symbol-list of length N. For example, if there is a 2-level factor in the design, then **H** must contain a 2-part symbol-list (call it **K**). The values in **K** will be used to represent the levels for all 2-level factors in the design. The length of **H** must equal the number of distinct level counts found in all the factors of the design. And **H** must be ordered by the distinct level counts. For example, if every factor in the design is either a 2-level or a 3-level factor and there is at least one of each type, then the length of **H** must be 2. The first symbol-list in **H** must be of length 2 and the second symbol-list in **H** must be of length 3.

For example, suppose you want to represent the “low” setting of a 2-level factor by the Alpha character and the “high” setting of a 2-level factor by the Beta character. Meanwhile, for 3-level factors, you want to represent “low”, “middle,” and “high” values by the Club, Diamond, and Heart characters, respectively. In this situation, **H** would be the list: `{{"α","β"},{"§","¨","©"}}`.

Evaluating the **DOEFactorial** function as shown:

```
factorialxdsn=DOEFactorial[<|"X1"->{0.7,1.9}, "X2"->{40,50,60}, "X3"->{1.8,2.3}|>,
RandomizeRuns->False,
DataRepresentation->{{"α","β"},{"♣","♢","♡"}}];
factorialxdsn[[1]]
```

will produce the following design:

	Pattern	X1	X2	X3
1	111	α	♣	α
2	211	β	♣	α
3	121	α	◊	α
4	221	β	◊	α
5	131	α	▽	α
6	231	β	▽	α
7	112	α	♣	β
8	212	β	♣	β
9	122	α	◊	β
10	222	β	◊	β
11	132	α	▽	β
12	232	β	▽	β

Colors can be used as symbols. For example, suppose you want to represent the “low” setting of a 2-level factor by the Red and the “high” setting of a 2-level factor by Green. Meanwhile, for 3-level factors, you want to represent “low”, “middle,” and “high” values by Orange, Blue, and Purple, respectively.

Evaluating the **DOEFactorial** function as shown:

```
factorialxdsn=DOEFactorial[<|"X1"->{0.7,1.9}, "X2"->{40,50,60}, "X3"->{1.8,2.3}|>,
RandomizeRuns->False,
DataRepresentation->{{Red,Green},{Orange,Blue,Purple}}];
factorialxdsn[[1]]
```

will produce the following design:

	Pattern	X1	X2	X3
1	111			
2	211			
3	121			
4	221			
5	131			
6	231			
7	112			
8	212			
9	122			
10	222			
11	132			
12	232			

By default, any DOE Toolkit function will also create an additional column in the resulting Dataset named “Pattern”. Each row of the “Pattern” column contains a string that represents the pattern of factor levels for that row in the design. You can turn off the “Pattern” column by setting the **`PatternColumn`** option to False.

You can control the format of the strings in the “Pattern” column by using the **`PatternRepresentation`** option. By default, the **`PatternRepresentation`** option is set to **`PatternsAreLevelListIndices`** and the strings in the “Pattern” column consist of concatenated level indices for each run.

When the **`PatternRepresentation`** option is set to **`PatternsAreMinusZeroPlus`**, the strings in the “Pattern” column obey the following rules: if every factor in the design has 2 levels, a given row of the “Pattern” column will display a string of “-“ and/or “+” signs. If some factor has 3 levels, a given row of the “Pattern” column will display a string of “-“ and/or “+” signs and/or zeros. If some factor has only 1 level or more than 3 levels, a given row of the “Pattern” column will display a string of “-“ and/or “+” signs and/or zeros and/or indices of the levels of the factors in that row.

You can also set the **`PatternRepresentation`** to a customized list of symbol-lists, following the same rules as described for the case where the **`DataRepresentation`** option is set to a list of symbol-lists.

For example:

```
factorialxdsn=DOEFactorial[<|"X1"->{0.7,1.9}, "X2"->{40,50,60}, "X3"->{1.8,2.3}|>,
RandomizeRuns->False,
PatternRepresentation->{{Red,Green},{Orange,Blue,Purple}}];
factorialxdsn[[1]]
```

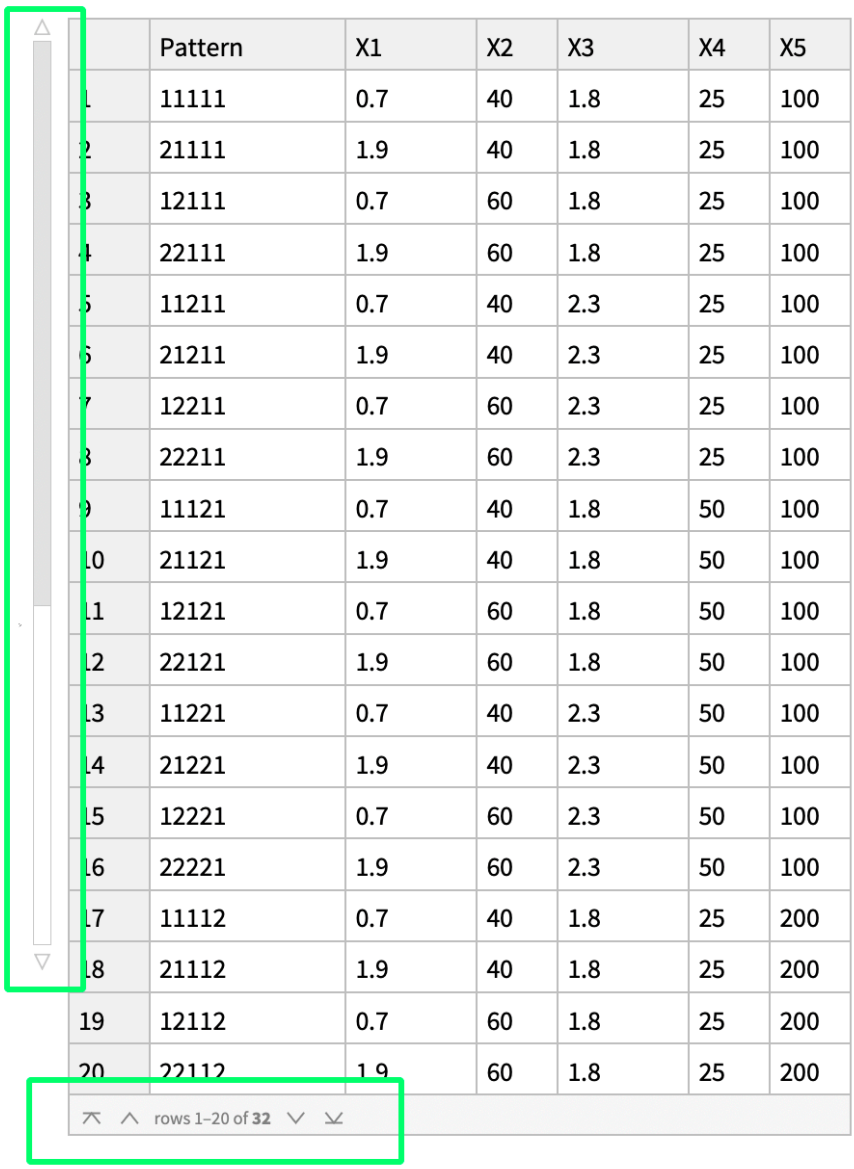
will produce the following design:

Row	Pattern	X1	X2	X3
1		0.7	40	1.8
2		1.9	40	1.8
3		0.7	50	1.8
4		1.9	50	1.8
5		0.7	60	1.8
6		1.9	60	1.8
7		0.7	40	2.3
8		1.9	40	2.3
9		0.7	50	2.3
10		1.9	50	2.3
11		0.7	60	2.3
12		1.9	60	2.3

Note that the **DOEMixture** function does not support either the **`DataRepresentation`** or **`PatternRepresentation`** option. Data in a **DOEMixture** design is always represented in terms of fractions. Patterns are represented either in terms of fraction lists or bar charts For more information, see the Mixture Designs section.

Unless you specify otherwise, Mathematica will determine how many rows of a Dataset object are displayed in your notebook. This means that for designs with several runs, Mathematica will only display a portion of the runs in the view it renders. In that case, Mathematica will provide a vertical scrollbar along the left side of the Dataset viewer. It will also provide chevrons (buttons) at the bottom-left of the Dataset viewer that allow you to navigate through the view of the design.

The next page gives an example in which the vertical scrollbar and bottom-left chevrons are highlighted.



	Pattern	X1	X2	X3	X4	X5
1	11111	0.7	40	1.8	25	100
2	21111	1.9	40	1.8	25	100
3	12111	0.7	60	1.8	25	100
4	22111	1.9	60	1.8	25	100
5	11211	0.7	40	2.3	25	100
6	21211	1.9	40	2.3	25	100
7	12211	0.7	60	2.3	25	100
8	22211	1.9	60	2.3	25	100
9	11121	0.7	40	1.8	50	100
10	21121	1.9	40	1.8	50	100
11	12121	0.7	60	1.8	50	100
12	22121	1.9	60	1.8	50	100
13	11221	0.7	40	2.3	50	100
14	21221	1.9	40	2.3	50	100
15	12221	0.7	60	2.3	50	100
16	22221	1.9	60	2.3	50	100
17	11112	0.7	40	1.8	25	200
18	21112	1.9	40	1.8	25	200
19	12112	0.7	60	1.8	25	200
20	22112	1.9	60	1.8	25	200

rows 1–20 of **32**

If you know the total number of runs in the design and you want the Dataset viewer to display them all, then you should specify the **`DatasetOptions`** option when you evaluate a DOE Toolkit function. Within the list of options you supply to the **`DatasetOptions`** option, you can specify the **`MaxItems`** option (which is a Dataset option). For example, if the total number of runs in the design is less than or equal to 64, setting the **`MaxItems`** option via the **`DatasetOptions`** option as shown below will cause the Dataset viewer to display all rows:

```
factorsAndLevels = <|"X1" -> {0.7, 1.9}, "X2" -> {40, 60}, "X3" -> {1.8, 2.3},
"X4" -> {25, 50}, “X5" -> {100, 200}|>;
factorialxdsn = DOEFactorial[factorsAndLevels,
RandomizeRuns->False, DatasetOptions->{MaxItems->64}];
factorialxdsn[[1]]
```

	Pattern	X1	X2	X3	X4	X5
1	11111	0.7	40	1.8	25	100
2	21111	1.9	40	1.8	25	100
3	12111	0.7	60	1.8	25	100
4	22111	1.9	60	1.8	25	100
5	11211	0.7	40	2.3	25	100
6	21211	1.9	40	2.3	25	100
7	12211	0.7	60	2.3	25	100
8	22211	1.9	60	2.3	25	100
9	11121	0.7	40	1.8	50	100
10	21121	1.9	40	1.8	50	100
11	12121	0.7	60	1.8	50	100
12	22121	1.9	60	1.8	50	100
13	11221	0.7	40	2.3	50	100
14	21221	1.9	40	2.3	50	100
15	12221	0.7	60	2.3	50	100
16	22221	1.9	60	2.3	50	100
17	11112	0.7	40	1.8	25	200
18	21112	1.9	40	1.8	25	200
19	12112	0.7	60	1.8	25	200
20	22112	1.9	60	1.8	25	200
21	11212	0.7	40	2.3	25	200
22	21212	1.9	40	2.3	25	200
23	12212	0.7	60	2.3	25	200
24	22212	1.9	60	2.3	25	200
25	11122	0.7	40	1.8	50	200
26	21122	1.9	40	1.8	50	200
27	12122	0.7	60	1.8	50	200
28	22122	1.9	60	1.8	50	200
29	11222	0.7	40	2.3	50	200
30	21222	1.9	40	2.3	50	200
31	12222	0.7	60	2.3	50	200
32	22222	1.9	60	2.3	50	200

Common Options for DOE Toolkit Functions

The following DOE Toolkit functions share several options in common:

DOEAugment
DOEBoxBehnken
DOECentralComposite
DOEFactorial
DOEFractional
DOEMixture
DOEOptimal
DOEOrthogonal

We have already described the **`DataRepresentation`** and **`PatternRepresentation`** options in the Common Features of DOE Toolkit Functions section. In this section we discuss some of the other options that are supported by all DOE Toolkit functions (with the exception of DOEDesirability).

Blocking a Design

In practice is common to use Randomized Block Design in order to eliminate the effects of so-called nuisance (aka blocking) factors in an experiment. The basic concept is to create homogeneous blocks within a design in which the blocking factors are held constant and the factor(s) of interest is/ are allowed to vary.

You can think of a randomized block experiment as a collection of completely randomized experiments, each run within one of the blocks of the total experiment.

It is very straightforward to generate a randomized block design for any function in the DOE Toolkit. You simply specify the **`DesignBlocking`** option, which points to a list of rules of the form:

BFactor->BLevelList.

The LHS of each rule (BFactor) provides the name to be assigned to the blocking factor. The RHS of each rule (BLevelList) is the list of levels associated with BFactor.

For example: **`DesignBlocking`**`->{"Operator"->{1,2,3}}` specifies that "Operator" is to be used as a blocking factor with 3 levels: {1,2,3}.

Suppose you are performing an agricultural experiment that involves 3 different fertilizers {X1,X2,X3}, each of which will be applied at 2 different levels. (For example, X1 will be applied at 5 ounces and 10 ounces, X2 will be applied at 4 and 12 ounces, and X3 will be applied at 8 and 16 ounces.) You want to create a full factorial design for the experiment. In order to conduct the experiment, 2 plots of land are used where the 3 fertilizers will be applied to the crop on each plot. To eliminate the effect

that a particular plot may have on the outcome of the experiment, we introduce a new (blocking) factor called “Plot” which has 2 levels {1,2} for the 2 plots involved.

Evaluating the **DOEFactorial** function as shown below will generate the desired randomized 2 block, full factorial design that follows.

```
factorsAndLevels = <|"X1" -> {5,10}, "X2" -> {4,12}, "X3" -> {8,16}|>;
factorialxdsn = DOEFactorial[factorsAndLevels,
DesignBlocking->{"Plot"->{1,2}}];
factorialxdsn[[1]]
```

	Pattern	X1	X2	X3	Plot
1	111	5	4	8	1
2	211	10	4	8	1
3	121	5	12	8	1
4	221	10	12	8	1
5	112	5	4	16	1
6	212	10	4	16	1
7	122	5	12	16	1
8	222	10	12	16	1
9	111	5	4	8	2
10	211	10	4	8	2
11	121	5	12	8	2
12	221	10	12	8	2
13	112	5	4	16	2
14	212	10	4	16	2
15	122	5	12	16	2
16	222	10	12	16	2

By default, the **DesignBlocksAreRows** option is set to True. When the **DesignBlocksAreRows** option is set to True, a column for each blocking factor is added to the Dataset and the runs corresponding to a given combination of levels of the blocking factors (aka blocks) are added as rows to the design.

When the **DesignBlocksAreRows** option is set to True, as rows are added for each block, the rows are alternately colored with two different colors. You can control these colors with the **DesignBlockColors** option. The **DesignBlockColors** option should be set to a list of two colors. By default, the **DesignBlockColors** option is set to the list {**LightBlue**, **White**}. In the example above which has 2 blocks, the first 8 rows, corresponding to the first block, are colored light blue and the second 8 rows, corresponding to the second block, are colored white. Here is the same

design but with the **DesignBlockColors** option set as: **DesignBlockColors->{ LightRed,Orange}:**

```
factorsAndLevels = <|"X1" -> {5,10}, "X2" -> {4,12}, "X3" -> {8,16}|>;
factorialxdsn = DOEFactorial[factorsAndLevels,
DesignBlocking->{"Plot"->{1,2}},
DesignBlockColors->{LightRed,Orange}];
factorialxdsn[[1]]
```

	Pattern	X1	X2	X3	Plot
1	211	10	4	8	1
2	222	10	12	16	1
3	122	5	12	16	1
4	112	5	4	16	1
5	121	5	12	8	1
6	111	5	4	8	1
7	221	10	12	8	1
8	212	10	4	16	1
9	111	5	4	8	2
10	121	5	12	8	2
11	222	10	12	16	2
12	122	5	12	16	2
13	112	5	4	16	2
14	211	10	4	8	2
15	221	10	12	8	2
16	212	10	4	16	2

If you set the **RandomizeRuns** option to True, then as a block is added to the design, the order of its runs is randomized. By default, the **RandomizeRuns** option is set to True. In the example above, the rows within each block have been randomized.

Meanwhile, if the **DesignBlocksAreRows** option is set to False, then the **DesignBlockColors** option is ignored. Columns are added to the Dataset such that there is a column for each combination of levels of the blocking factors and no rows are added to the design. If you don't need to randomize the runs within each block, then you may find this is a more convenient way to work with the design. For example, here is the design that results with the **DesignBlocksAreRows** option set to False for the same 2-level "Plot" blocking factor:

```
factorsAndLevels = <|"X1" -> {5,10}, "X2" -> {4,12}, "X3" -> {8,16}|>;
factorialxdsn = DOEFactorial[factorsAndLevels,
DesignBlocking->{"Plot"->{1,2}},
DesignBlocksAreRows->False,
DesignBlockColors->{Green,Yellow}];
factorialxdsn[[1]]
```

	Pattern	X1	X2	X3	Plot=1	Plot=2
1	211	10	4	8	0	0
2	221	10	12	8	0	0
3	111	5	4	8	0	0
4	222	10	12	16	0	0
5	212	10	4	16	0	0
6	122	5	12	16	0	0
7	112	5	4	16	0	0
8	121	5	12	8	0	0

Blocking by Aliasing and Incomplete Block Designs: See the Fractional Factorial Designs section for information on these topics.

Other Common Options

You can add central points to any design (with the exception of designs generated by the **DOEMixture** function) by using the **`CentralPoints`** option.

You assign a list of *insertion rules* of the form LHS -> RHS to the **`CentralPoints`** option. The LHS of the rule is either one of the keywords: **`{DesignInsertAtTop, DesignInsertAtBottom, DesignInsertInMiddle}`** or else a row number, which indicates the insertion location. The RHS of the rule indicates the number of central points to be inserted at the designated location. For example:

```
factorialxdsn = DOEFactorial[<|"X1"->{0.7, 1.7, 2.7}, "X2"->{40, 50, 60}, “X3"->{1.8, 2.3,2.8}|>,
RandomizeRuns->False,
CentralPoints->{DesignInsertAtTop->1, DesignInsertAtBottom->3,
DesignInsertInMiddle->2,4->1}];
factorialxdsn[[1]]
```

The result is shown below:

	Pattern	X1	X2	X3
1	○○○	1.7	50	2.3
2	111	0.7	40	1.8
3	211	2.7	40	1.8
4	121	0.7	60	1.8
5	○○○	1.7	50	2.3
6	221	2.7	60	1.8
7	112	0.7	40	2.3
8	○○○	1.7	50	2.3
9	○○○	1.7	50	2.3
10	212	2.7	40	2.3
11	122	0.7	60	2.3
12	222	2.7	60	2.3
13	113	0.7	40	2.8
14	213	2.7	40	2.8
15	123	0.7	60	2.8
16	223	2.7	60	2.8
17	○○○	1.7	50	2.3
18	○○○	1.7	50	2.3
19	○○○	1.7	50	2.3

In the example above, 1 central point was inserted at the top of the design, 3 were added to the bottom, 2 were added to the middle, and 1 was added at row 4 (relative to the original design, prior to adding any central points).

The **`CentralPoints`** option supports the **`NumberPerFactor`** keyword when specifying the number of central points to be inserted. For example, the code below will generate a design in which 6 central points are inserted at the top:

```
factorialxdsn = DOEFactorial[<|"X1" -> {0.7, 1.9}, "X2" -> {40, 50, 60}, "X3" -> {1.8, 2.3}|>,
RandomizeRuns->False,
CentralPoints->{DesignInsertAtTop ->NumberPerFactor->2}];
factorialxdsn[[1]]
```

Similarly, you can add axial (aka *star*) points to any design (with the exception of designs generated by the **DOEMixture** function) by using the **`AxialPointGroups`** option. You assign a 2-part list to the **`AxialPointGroups`** option. The first part is either a number or an *alpha-type* from the list:

{DesignAlphaRotatable, DesignAlphaOrthogonal, DesignAlphaSpherical}. The second part is a list of *insertion rules* exactly the same as those used by the **CentralPoints** option. If you set Part 1 to one of the *alpha-types*, then the value of alpha will be computed based on a formula that takes into account some or all of the following properties of the design: number of factors, number of levels per factor, number of central points. The *alpha-types* **DesignAlphaRotatable** and **DesignAlphaOrthogonal** only be used when the number of levels per factor is a constant.

Since axial points are always inserted as a group (2 rows are added to the design for each factor), the number you specify *as* the RHS of an *insertion rule* in the **AxialPointGroups** option is the *number of groups* to be inserted.

For example:

```
factorialxdsn = DOEFactorial[<|"X1" -> {0.7, 2.7}, "X2" -> {40, 50, 60}, "X3"->{1.8, 2.3, 2.8}|>,
RandomizeRuns->False,
AxialPointGroups->{Orthogonal,{Bottom->1}}];
factorialxdsn[[1]]
```

The result is shown below. One group of axial points (6 rows) is inserted at the bottom of the design:

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	2.7	40	1.8
3	121	0.7	60	1.8
4	221	2.7	60	1.8
5	112	0.7	40	2.3
6	212	2.7	40	2.3
7	122	0.7	60	2.3
8	222	2.7	60	2.3
9	113	0.7	40	2.8
10	213	2.7	40	2.8
11	123	0.7	60	2.8
12	223	2.7	60	2.8
13	{-α, 0, 0}	-0.0320508	50	2.3
14	{+α, 0, 0}	3.43205	50	2.3
15	{0, -α, 0}	1.7	32.6795	2.3
16	{0, +α, 0}	1.7	67.3205	2.3
17	{0, 0, -α}	1.7	50	1.43397
18	{0, 0, +α}	1.7	50	3.16603

You can add your own metadata to the design generated by a DOE Toolkit function by specifying the **`MyMetadata`** option. This option must be set to a list of rules. The LHS of each rule can be any string. For example:

```
factorialxdsn = DOEFactorial[<|"X1" -> {0.7, 2.7}, "X2" -> {40, 60}, "X3" -> {1.8, 2.3, 2.8}|>,
"MyMetadata"->{"Description"->"This full factorial design was generated 09-23-2025."}];
factorialxdsn[[1]]
factorialxdsn[[2]]//TableForm
```

You can set various properties of the Dataset object returned by a DOE Toolkit function by specifying the **`DatasetOptions`** option. This option must be set to a list of options that are supported by Mathematica's Dataset object. The Common Features of DOE Toolkit Functions section shows how to could control the number of rows displayed in the Dataset object by passing the **`MaxItems`** Dataset option to the Dataset object via the **`DatasetOptions`** option. In the example below, we use the **`DatasetOptions`** option to pass along some header options to the Dataset object:

```
factorialxdsn = DOEFactorial[<|"X1" -> {0.7, 2.7}, "X2" -> {40, 60}, "X3" -> {1.8, 2.3, 2.8}|>,
RandomizeRuns->False,
DatasetOptions->{HeaderBackground->{LightGreen,Cyan}, HeaderStyle->Bold}];
factorialxdsn[[1]]
```

The headers of the Dataset object are rendered in bold font. The background of the column of row numbers is light green and the background of the column names is cyan:

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	2.7	40	1.8
3	121	0.7	60	1.8
4	221	2.7	60	1.8
5	112	0.7	40	2.3
6	212	2.7	40	2.3
7	122	0.7	60	2.3
8	222	2.7	60	2.3
9	113	0.7	40	2.8
10	213	2.7	40	2.8
11	123	0.7	60	2.8
12	223	2.7	60	2.8

You can add extra columns to the design generated by a DOE Toolkit function by specifying the **`ExtraColumns`** option. This option must be set to a list. Each element ***e*** of the list must be either a string or a 2-part list. If the element ***e*** is a string *s*, then *s* is used as the name of the column and the initial value of each cell in the column is set to Missing["Nonexistent"]. If the element **e** is a 2-part list {*s, n*}, then *s* is used as the name of the column and *n* is used as the initial value for the cells in the column.

For example:

```
factorialxdsn = DOEFactorial[<|"X1" -> {0.7, 1.7}, "X2" -> {40, 60}, "X3" -> {1.8, 2.8}|>,
RandomizeRuns->False,
ExtraColumns->{"Response"}];
factorialxdsn[[1]]
```

The result is shown below. An additional column named "Response" is added to the design matrix:

	Pattern	X1	X2	X3	Response
1	211	1.7	40	1.8	—
2	221	1.7	60	1.8	—
3	122	0.7	60	2.8	—
4	212	1.7	40	2.8	—
5	112	0.7	40	2.8	—
6	222	1.7	60	2.8	—
7	111	0.7	40	1.8	—
8	121	0.7	60	1.8	—

Consult Appendix 2: Common Options for a complete list of all common DOE Toolkit options.

Box-Behnken Designs

A Box-Behnken design is an independent quadratic design in that it does not contain an embedded factorial or fractional factorial design. In this design the treatment combinations are at the midpoints of the edges of the process space and at the center. These designs are rotatable (or near rotatable) and require 3 levels of each factor. The designs have limited capability for orthogonal blocking compared to central composite designs.

The DOE Toolkit defines the **DOEBoxBehnken** function for creating Box-Behnken designs.

The **DOEBoxBehnken** function is restricted to designs with at least 3 factors but no more than 7 factors. Each factor must have exactly 3 levels. If your model has more than 7 factors, we recommend that you try a Central Composite design instead.

The **DOEBoxBehnken** function supports all the options that are described in Appendix 2: Common Options.

There are no options unique to the **DOEBoxBehnken** function.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for an example of using the **DOEBoxBehnken** function.

Central Composite Designs

A Central Composite Design (aka Box-Wilson Central Composite Design) contains an imbedded factorial or fractional factorial design with center points that is augmented with a group of so-called "star points" that allow estimation of curvature. If the distance from the center of the design space to a factorial point is ±1 unit for each factor, then the distance (α) from the center of the design space to a star point is such that |α| > 1. The precise value of α depends on certain properties desired for the design and on the number of factors involved.

The DOE Toolkit defines the **DOECentralComposite** function for creating central composite designs.

There are 3 different "classic" forms of central composite design: Central Composite Circumscribed (CCC), Central Composite Inscribed (CCI), and Central Composite Face Centered (CCF). You can construct any of the three classic forms using the **DOECentralComposite** function. Historically, CCC and CCI designs require 5 levels of each factor. Meanwhile, CCF designs require 3 levels of each factor. However, the **DOECentralComposite** function does not enforce these restrictions so you can use 2-level, 4-level, etc. factors if so desired.

You can set α to a specific number or to one of the keywords: {**DesignAlphaRotatable, DesignAlphaOrthogonal, DesignAlphaSpherical**}. When α is set to one of the keywords it is computed as follows:

Let k=number of factors, F=number of runs in full factorial (or fractional factorial) part of the design, and n=number of central points.

Orthogonal	α = (Q*F/4)^(1/4) where: T = 2*k + n Q = (Sqrt[F+T]-Sqrt[F])^2
Rotatable	α = F^(1/4)
Spherical	α = Sqrt[k]

The **DOECentralComposite** function supports all the options that are described in Appendix 2: Common Options. . In addition, the following options provided by the **DOEFractional** function are supported:

ConfoundingRules,
UseInteractionTables,
InteractionTable2x2,
InteractionTable3x3,
InteractionTable2x3

Thus, to reduce the number of runs that would otherwise be required for a full factorial design, you can use the **ConfoundingRules** option to alias certain factors with others.

For more information about these options, see the documentation for the **DOEFractional** function in Appendix 1: DOE Toolkit Function Summary.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for some examples of using the **DOECentralComposite** function.

Desirability Manipulator

The DOE Toolkit includes the **DOEDesirability** function, which allows you to explore the desirability of your single/multiple response model for alternative settings of the independent variables in the model.

The **DOEDesirability** function provides slider controls that allow you to adjust the values of the independent variables in your model. Manipulating the sliders causes the desirability function for each response variable to be recomputed and its associated ListPlots to be redrawn. This allows you to interactively explore the desirability of the multiple response model for a given set of factor settings.

Let's assume that you have used one of the DOE Toolkit functions to create an experimental design and that you have run your experiment and recorded the results. In order to be able to use the **DOEDesirability** function to analyze the desirability of the factor settings, you must first complete the following steps:

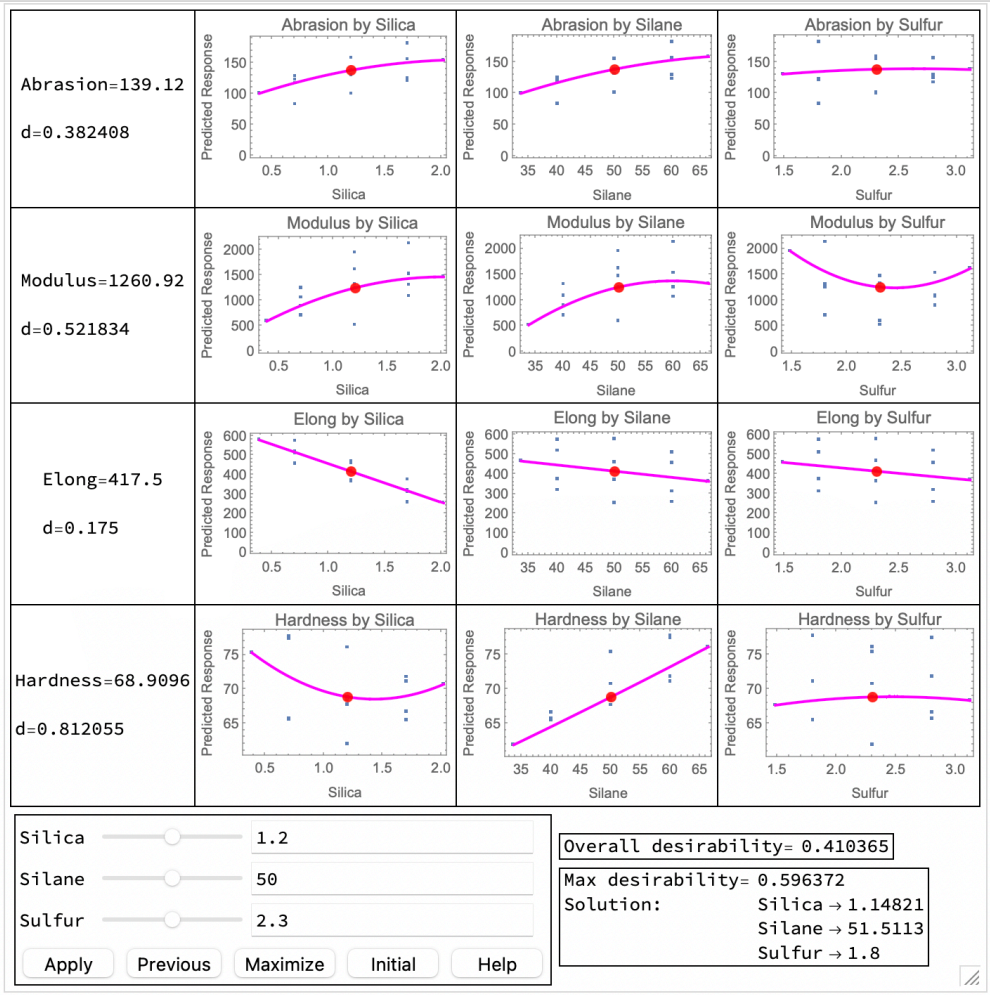
(1) Define vectors for the observed effects in your model. These vectors are the observed values for the effect variables that were obtained by running your experiment.
(2) Define vectors (derived from your design) for the factors in your model.
(3) Create a fitted model for each of the effects in your model.
(4) Define a list of the fitted models created in Step 3.
(5) Define the effect-based desirability functions to be used for your model.
(6) Set lower, target, upper values for the effects in your model.
(7) Define factor-based desirability functions.
(8) Define a list of the factor-based desirability functions created in Step 7.

The **DOE Toolkit User Guide 3.0.nb** notebook provides an example in which these steps are followed in order to use the **DOEDesirability** function to analyze a multiple response model based on the example described in Derringer and Suich[3]. (See Appendix 5: References for more information.)

The screen capture on the following page shows the **DOEDesirability** function in use for the D&S model.

Evaluate the DOEDesirability function and use its slider controls to explore the desirability of alternative factor settings.

```
DOEDesirability[fittedModelList, desireFcnListDS, {silica, silane, sulfur},
factorNames, {abrasion, modulus, elong, hardness}, effectNames,
 MinMaxForXVars → factorLevelLists, FitPlotColor → Magenta]
```



The **DOEDesirability** function in use for the D&S model

Full Factorial Designs

A Full Factorial Design is one in which every setting of every factor appears with every setting of every other factor.

The DOE Toolkit defines the **DOEFactorial** function for creating full factorial designs.

The **DOEFactorial** function supports all the options that are described in Appendix 2: Common Options.

There are no options unique to the **DOEFactorial** function.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for some examples of using the **DOEFactorial** function.

Fractional Factorial Designs

The ASQC (1983) Glossary & Tables for Statistical Quality Control defines a Fractional Factorial Design in the following way: "A factorial experiment in which only an adequately chosen fraction of the treatment combinations required for the complete factorial experiment is selected to be run."

The DOE Toolkit defines the **DOEFractional** function for creating fractional factorial designs.

The **DOEFractional** function allows you to reduce the number of runs your design would otherwise require by enabling you to confound (aka *alias*) some factors to other factors. For example, suppose you have 3 factors named "X1", "X2", "X3", each of which is a 3-level factor. Ordinarily, a full factorial design would require 27 runs for such a model. However, using **DOEFractional**, you could confound X3 with X1 and X2, and thereby reduce the number of runs to 9. To define the aliasing structure, you would use the **`ConfoundingRules`** option to specify the confounding rule `{"X3"->{"X1","X2"}}`.

The **DOEFractional** function supports all the options that are described in Appendix 2: Common Options. In addition, the following options are supported:

`ConfoundingRules`->conf, where conf is set to a list of confounding rules. Each confounding rule has the form: ConfVar->{Var1,Var2,…,VarK} where ConfVar is the name of a factor and {Var1,Var2,…,VarK} are the names of the factors with which ConfVar is to be confounded. If conf is the empty list then the **DOEFractional** function computes the same design as the **DOEFactorial** function.

By default, when two factors interact and each factor has either 2 or 3 levels, one of the tables assigned to the options: **`InteractionTable2x2`**, **`InteractionTable2x3`**, or **`InteractionTable3x3`** is used to compute the value of the interaction. You can turn off this default behavior by setting the **`UseInteractionTables`** option to False. If the **`UseInteractionTables`** option is False, then the product of the level values of the factors involved is used to represent the value of the interaction.

`UseInteractionTables`->flag, where flag is either True or False. If flag is set to True, then one of the tables assigned to the options: **`InteractionTable2x2`**, **`InteractionTable2x3`**, or **`InteractionTable3x3`** is used to compute the value of the interaction for two factors each of which has either 2 or 3 levels. By default, flag is set to True.

Consult the definition of the **DOEFractional** function in Appendix 1: DOE Toolkit Function Summary for the default definitions of the interaction table options.

The *Resolution* of a design describes the degree to which estimated main effects are aliased (confounded) with 2-level, 3-level, etc. interactions. When you evaluate the **DOEFractional** function to generate a design, the Resolution of your design is controlled by the list of confounding rules that is specified by the **`ConfoundingRules`** option.

For example, if you want a 2_{III}^{3-1} design, then the third factor in your design should be confounded with the first two:

`ConfoundingRules``->{"X3"->{"X1","X2"}}`

Likewise, if you want a 2_{IV}^{4-1} design, then the fourth factor in your design should be confounded with the first three:

`ConfoundingRules``->{"X4"->{"X1","X2","X3"}}`

Meanwhile, if you have 8 factors a design with resolution 2_{IV}^{3-3} is generated using the following confounding rules:

```
ConfoundingRules->{"X6"->{"X1","X2","X3"},
                   "X7"->{"X1","X2","X4"},"},
                   "X8"->{"X2","X3","X4","X5"}}
```

An excellent discussion of design Resolution can be found at the **National Institute of Standards and Technology** (aka NIST) website in their **Engineering Statistics Handbook**[4]. Cf. section 5.3.3.4.4.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for some examples of using the **DOEFractional** function.

Mixture Designs

In a Mixture Design experiment, the independent factors are proportions of different components of a blend.

The DOE Toolkit defines the **DOEMixture** function for creating Mixture Designs.

The **DOEMixture** function supports only *standard mixture designs* at the present time. Thus, the mixture components are subject to the constraint that they must sum to one. So-called "*constrained mixture designs*" or "*Extreme-Vertices*" designs are not supported in this release.

The **DOEMixture** function supports most of the options that are described in Appendix 2: Common Options. However, it does not support the following options:

DataRepresentation,
PatternRepresentation,
CentralPoints,
AxialPointGroups

In addition, the following options are supported:

MixtureType->mixtype, where mixtype is set one of the keywords in {**SimplexLattice, SimplexCentroid**}. By default, mixtype is set to **SimplexLattice**.

MixtureLevelsPerFactor->nlev, where nlev is either the keyword **MixtureNumberFactors** or else an integer >= 2. By default, nlev is set to **MixtureNumberFactors**.

PatternIsBarChart->flag, where flag is either True or False. If flag is set to True, then the Pattern column displays a bar chart for each row of the design to represent the pattern for that row of the design. Otherwise, the pattern for the row is displayed as a list of fractions. By default, flag is set to True. This option is ignored if the **PatternColumn** option is set to False.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for some examples of using the **DOEMixture** function.

Orthogonal Designs

In a balanced, Orthogonal Design, the vectors of level values for the independent factors are pair-wise orthogonal and each level of each factor appears the same number of times in each factor.

The DOE Toolkit defines the **DOEOrthogonal** function for creating balanced, orthogonal designs. Many of the designs that the **DOEOrthogonal** function generates are identical to what are sometimes called Taguchi Arrays.

The **DOEOrthogonal** function is restricted to designs in which each factor is either 2-level or 3-level.

The **DOEOrthogonal** function supports all the options that are described in Appendix 2: Common Options.

In addition, the following options are supported:

`NumberIterations`->numIter, where numIter is an integer >= 1. By default, numIter is set to 10. If the **DOEOrthogonal** function has failed to find an orthogonal design by the time numIter iterations have been performed, the **DOEOrthogonal** function will halt. An iteration occurs for each attempt to find a mutually orthogonal set of factor vectors for a given run-length.

The **DOEOrthogonal** function allows you to cancel the computation while it is in progress: If you are running the Windows or Linux operating system, hold down the **CTRL** (aka **Control**) key until you see a message in your notebook to the effect that the operation has been aborted. If you are on a Macintosh, hold down the **Option** key.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for some examples of using the **DOEOrthogonal** function.

Optimal Designs

Optimal Designs result from either maximizing or minimizing some optimality criterion relative to a subset of the runs in some pre-existing design.

The DOE Toolkit defines the **DOEOptimal** function for creating optimal designs.

The **DOEOptimal** function creates an optimal experimental design with the number of runs you specify, based on the XDesign object that you provide as input. The design that is generated is returned as part of an XDesign object.

The **DOEOptimal** function can create D-optimal, A-optimal, T-optimal, and G-optimal designs.

Example of output produced while **DOEOptimal** is running along with log information.

```
factors={X1, X2, X3, X4}
levelLists={{0.7, 1.7}, {40, 60}, {1.8, 2.8}, {100, 150}}
getRepresentation: (DataRepresentation) rep=DataAreLevelListValues
dataRepresentation=DataAreLevelListValues
getRepresentation: (PatternRepresentation) rep=PatternsAreMinusZeroPlus
patternRepresentation=PatternsAreMinusZeroPlus
repLevelIndicesPerFactor={{1, 2}, {1, 2}, {1, 2}, {1, 2}}
Method: RowExchange
OptimalityCriterion: D (Maximize det(X'X))
time taken so far=00:00:04.5182127
estimated time remaining=00:00:08.2247348
n row exchanges made: 540
Hold down SHIFT key to abort.
Starting indices chosen at random, current iteration={1, 2, 3, 4, 7, 8, 9, 12, 14, 15}
nImprovementsThisIteration=8 n row exchanges=120
d1CurIteration=5.42592×10^9 after iteration=1
***Optimality criterion was improved
Starting indices chosen at random, current iteration={3, 4, 5, 7, 8, 9, 10, 11, 12, 16}
nImprovementsThisIteration=9 n row exchanges=120
d1CurIteration=5.42592×10^9 after iteration=2
No improvement in optimality criterion
Starting indices chosen at random, current iteration={1, 2, 3, 4, 5, 6, 8, 11, 13, 15}
nImprovementsThisIteration=9 n row exchanges=120
d1CurIteration=5.42592×10^9 after iteration=3
```

The **DOEOptimal** function supports all the options that are described in Appendix 2: Common Options. In addition, the following options are supported:

`OptimalityCriterion`->crit, where crit is set one of the keywords in **{`OptimalA, OptimalD, OptimalG, OptimalT`}.** By default, crit is set to **`OptimalD`**.

`OptimalityMethod`->methd, where methd is set one of the keywords in {**`CoordinateExchange`, `RowExchange`, `Exhaustive`**}. By default, methd is set to **`RowExchange`**.

`NumberIterations`->nIter, where nIter is an integer >= 1. By default, nIter is set to 10. This option is ignored if the **`OptimalityMethod`** option is set to **`Exhaustive`**.

`StartingIndices`->startInd, where startInd is a list containing nRuns distinct integers between 1 and N, where N is the number of runs in the Dataset component of the input XDesign object. By default, startInd is set to the empty list.

`ShowBestCandidates`->flag, where flag is either True or False. If flag is set to True, then the list of best candidates discovered in the course of searching for an optimal design will be printed to your notebook. By default, flag is set to False.

In order to use the **DOEOptimal** function, you must first create an XDesign object, which you will provide as the first input parameter to the function. You can either create the XDesign object manually, or you can use the XDesign object returned by some other DOE Toolkit function. For example, you could use the XDesign object returned by the **DOEFactorial** function as the xdsnIn input parameter to the **DOEOptimal** function.

If you use a DOE Toolkit function to create an XDesign object (call it *xdsn*) that will be used as input to the **DOEOptimal** function, the **`DataRepresentation`** used by *xdsn* must be set to **`DataAreLevelListValues`**. In other words, when a DOE Toolkit function such as **DOEFactorial** is evaluated to generate *xdsn*, **DataRepresentation->DataAreLevelListValues** should be provided as an option to whatever DOE Toolkit function was used to generate *xdsn.*

The **DOEOptimal** function is different from the other functions that the DOE Toolkit provides. The function is dynamic: it provides feedback to the user by displaying a progress indicator as well as a continually updated estimate of the time required for its search algorithm to complete. Because the search for an optimal design may take longer than desired, you can always terminate the operation by selecting the **Abort Evaluation** command from the **Evaluation** menu. However, any results that the **DOEOptimal** function may have obtained up to that point will be lost. To remedy this problem, the **DOEOptimal** function provides another mechanism that allows you to cancel the computation while it is in progress: If you are running the Windows or Linux operating system, hold down the **CTRL** (aka **Control**) key until you see a message in your notebook to the effect that the operation has been

aborted. If you are on a Macintosh, hold down the **Option** key. If you terminate the operation in this way, the **DOEOptimal** function will display the best candidate it found (if any) prior to the time you terminated the operation. In some cases, this may give you a design that is good enough for your purposes.

If it is successful, the **DOEOptimal** function will create an XDesign object that encapsulates an optimal design for the criterion you specified.

Unless you use the **`StartingIndices`** option to specify a set of indices to define an initial design, the search for an optimal design begins with an initial design selected at random. Due to the nature of the algorithm involved, it is possible that the search for a solution will only find a local optimum. You can specify the **`NumberIterations`** option to run the **DOEOptimal** function as many times as desired to increase the probability that the algorithm will find a "near-best" solution. By default, the **`NumberIterations`** option is set to 10.

The **DOEOptimal** function provides the **`OptimalityMethod`** option which allows you to choose the algorithm to follow in the search for an optimal candidate. The choices for the **`OptimalityMethod`** option are:

`RowExchange`, (the default)
`CoordinateExchange`,
`Exhaustive`

The algorithm for the **`RowExchange`** method is based on Fedorov[2]. The algorithm for the **`CoordinateExchange`** method is based on Myer & Nachtsheim [1]. (See Appendix 5: References for more information.)

Please note: The **`CoordinateExchange`** method presupposes that all the factors in the input Dataset are independent of each other (i.e., no factors are functions of other factors, no factors are interactions with or confounded with other factors). If such dependencies exist, then you should only use the **`RowExchange`** method.

The **`Exhaustive`** method will try every possible candidate with the number of runs you requested, so you probably won't want to use this method unless the number of rows in your input Dataset is fairly small or you are willing to wait to obtain the result.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for some examples of using the **DOEOptimal** function.

Augmenting a Design

You can always add central points, axial points, replicates, randomization, and blocks to a design using the **DOEAugment** function. If there is a blocking factor in the design, then each block will receive the number of central at the locations specified by the **`CentralPoints`** option. Likewise, if there is a blocking factor in the design, then each block will receive the number of axial point groups at the locations specified by the **`AxialPointGroups`** option. In addition, the **DOEAugment** function allows you to add extra columns to your design.

When you evaluate the **DOEAugment** function and specify one or more options, you provide an XDesign object as input and the function *returns* an XDesign object with the modifications you requested. Thus, the **DOEAugment** function does not actually modify or in any other way change the XDesign object that you supply as input. Instead, it creates a *new* XDesign object based on your input. Although our descriptions in this document may sometimes suggest that the **DOEAugment** function is modifying the input Dataset object, it is important to keep in mind that strictly speaking, this is not the case.

The **DOEAugment** function supports the following subset of common options:

`CentralPoints`,
`AxialPointGroups`,
`AxialPointsAboveCentralPoints`,
`RandomizeRuns`,
`DesignBlocking`,
`DesignBlocksAreRows`,
`DesignBlockColors`,
`NumberReplicates`,
`MyMetadata`,
`RandomNumberSeed`,
`DatasetOptions`,
`ExtraColumns`
`PrintLog`

For more information about these options, see Appendix 2: Common Options.

Consult the **DOE Toolkit User Guide 3.0.nb** notebook for some examples of using the **DOEAugment** function with these options.

Creating Taguchi Designs

Many of the experimental designs that have been variously called **Taguchi Designs** or **Orthogonal Arrays** are actually fractional factorial designs. Those Taguchi designs which are fractional factorial in nature can be created using the **DOEFractional** function.

Taguchi designs for models in which all the factors have 2 levels are usually named using the construction: "Ln" where the letter "n" is replaced with some number (n is the number of runs in the design). For example, "L4" refers to a Taguchi design with 4 runs. Unfortunately, this naming convention is somewhat misleading; there can be more than one "Ln" design for a given number n. For example, there is an L8 design for 4 factors and there is also an L8 design for 5 factors. Although 8 runs are involved in each case, these two "L8" designs are not the same. Some Taguchi designs are actually full factorial; for example, L4 and L8 designs exist for a model with 3 factors. But in this case, the L8 design is actually a full factorial design (which can be created using the **DOEFactorial** function).

In most cases, there is more than one Taguchi design that can be chosen for a given number of factors. For example, in the case of a 5 factor model, one can choose from Taguchi designs L8, L12, and L16. Some Taguchi designs such as L12, are not, strictly speaking, fractional factorial designs and so cannot be produced using the **DOEFractional** function. Those Taguchi designs which are not fractional factorials can be produced using the **DOEOrthogonal** function.

The **DOE Toolkit User Guide 3.0.nb** notebook contains examples of how to produce Taguchi designs using the **DOEFractional** function for models with up to 6 factors each of which is 2-level.

The **DOEFractional** function can also be used to create most of the Taguchi designs for models in which all the factors are 3-level and for models in which some factors are 2-level while others are 3-level.

Obtaining Version Information

To display the version of the DOE Toolkit that is loaded in your notebook, evaluate the **DOEToolkitVersion**[] function like so:

```
DOEToolkitVersion[]
```

Reporting a Problem

To report a problem with the DOE Toolkit, send email to Harper Corditt Software at the following address:

harpercorditt@gmail.com

Be sure to include a description of your problem and please provide a Mathematica notebook that can be used to reproduce the problem. You can either attach the notebook to your email or else provide a link to the notebook in the Wolfram Cloud.

Please include your telephone number in case we need to contact you for further information.

Last but not least, please include your Mathematica license in the text of your email. To obtain your Mathematica license, follow these steps:

Step 1: Evaluate the following expression in your notebook:

```
$LicenseID
```

This should produce output similar to what you see below:

```
L1234-5678
```

Step 2: Select the string that is output and from the **Edit** menu, select the **Copy As** command, and from the **Copy As** submenu**,** select **Plain Text**.

Step 3: Paste the Mathematica license string into your email.

Getting Help

To obtain information about a particular **DOE Toolkit** function, click in your notebook to start a new Input cell. Then enter the "?" character followed by the name of the function and press the Enter key. For example, to obtain information about the **DOEFactorial** function you would enter the following:

```
?DOEFactorial
```

The following description is printed to your notebook:

Symbol

xdsn=DOEFactorial[*factorsAndLevels*] generates a full factorial experimental design for the factors described by *factorsAndLevels*. The function returns xdsn, a 2–part list. xdsn[[1]] is a Dataset object that encapsulates the generated design. xdsn[[2]] is a list of metadata for the generated design.

It may happen that rather than seeing a description such as the one shown above, you see something like this instead:

```
Missing[UnknownSymbol, doeFactorial]
```

When you see this message, there are two possible reasons why Mathematica considers the symbol in question to be Missing. First of all, it may be that you misspelled the name of the function, so be sure to check your typing. Remember that Mathematica is case sensitive. If you are sure that the name is not misspelled, then it is most likely the case that the DOE Toolkit is not loaded. Try evaluating the **Get** function, providing the path to the **DOEToolkit.wl** file as input to the **Get** function. (See the Loading the DOE Toolkit in Your Notebook section for more information.) Verify that there are no errors when evaluating the **Get** function for the **DOEToolkit.wl** file.

To obtain a list of all the options supported by a particular **DOE Toolkit** function, evaluate the Mathematica **Options** function, providing the name of the **DOE Toolkit** function as the input parameter to the **Options** function. For example, to see the list of options supported by the **DOEFactorial** function you would enter the following:

```
Options[DOEFactorial]
```

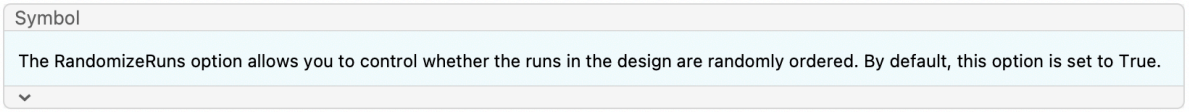
The following list is printed to your notebook:

```
{DataRepresentation → DataAreLevelListValues, PatternRepresentation → PatternsAreLevelListIndices,
 CentralPoints → {}, AxialPointGroups → {}, AxialPointsAboveCentralPoints → True,
 RandomizeRuns → True, DesignBlocking → {}, DesignBlocksAreRows → True, DesignBlockColors → {, },
 NumberReplicates → 0, PatternColumn → True, MyMetadata → {}, RowNumbers → True,
 RandomNumberSeed → BasedOnCurrentTime, DatasetOptions → {}, ExtraColumns → {}, PrintLog → False}
```

To obtain information about an option that is common to all DOE Toolkit functions, you enter the "?" character followed by the name of the option, followed by the Enter key. For example, to obtain information about the `RandomizeRuns` option, you would enter the following:

```
?RandomizeRuns
```

The following description is printed to your notebook:



Last but not least, you can always refer to Appendix 1: DOE Toolkit Function Summary of this document to obtain information about any DOE Toolkit function and its associated options.

Appendix 1: DOE Toolkit Function Summary

Function: **DOEAugment**

Usage:

xdsnAug = DOEAugment[*xdsnIn*] augments the design encapsulated by the Dataset object *xdsnIn*[[1]]. The function returns *xdsnAug*, a 2-part list. *xdsnAug*[[1]] is a Dataset object that encapsulates the augmented design. *xdsnAug*[[2]] is a list of metadata that contains the original metadata from *xdsnIn*[[2]] along with a record of the modifications that were made to augment the design.

The **DOEAugment** function supports the following subset of common options:

CentralPoints,
AxialPointGroups,
AxialPointsAboveCentralPoints,
RandomizeRuns,
DesignBlocking,
DesignBlocksAreRows,
DesignBlockColors,
NumberReplicates,
MyMetadata,
RandomNumberSeed,
DatasetOptions,
ExtraColumns,
PrintLog

These options are described in Appendix 2: Common Options.

Options Unique to DOEAugment:

(None)

Example:

```
xdsnAug = DOEAugment[xdsnIn,
   CentralPoints->{DesignInsertAtTop ->1, DesignInsertAtBottom ->1}];
```

Function: **DOEBoxBehnken**

Usage:

xdsn = DOEBoxBehnken[*factorsAndLevels*] generates a Box-Behnken experimental design for the factors described by *factorsAndLevels*. The input argument *factorsAndLevels* is an association, which connects the name of each factor to its level-values. The function returns xdsn, a 2-part list. xdsn[[1]] is a Dataset object that encapsulates the generated runs. xdsn[[2]] contains metadata.

DOEBoxBehnken supports designs with at least 3 factors but no more than 7 factors. Each factor must have exactly three levels.

This function supports all the options that are described in Appendix 2: Common Options.

Options Unique to DOEBoxBehnken:

(None)

Example:

```
boxbxdsn = DOEBoxBehnken[<|"X1" -> {0.7, 1.7, 2.7}, "X2" -> {40, 60, 80},
"X3" -> {1.8, 2.3, 2.8}|>]
```

Function: **DOECentralComposite**

Usage:

xdsn = DOECentralComposite[*factorsAndLevels*] generates a central composite experimental design for the factors described by *factorsAndLevels*. The input argument *factorsAndLevels* is an association, which connects the name of each factor to its level-values. The function returns xdsn, a 2-part list. xdsn[[1]] is a Dataset object that encapsulates the generated runs. xdsn[[2]] contains metadata.

Whenever the **DOECentralComposite** function is evaluated, the **AxialPointGroups** option is set by default to:

AxialPointGroups->{**DesignAlphaRotatable**, {**Bottom**->1}}.

In other words, if you don't specify the **AxialPointGroups** option at all, then **AxialPointGroups** ->{**DesignAlphaRotatable**, {**Bottom**->1}} is assumed.

Since axial points are always inserted as a group (2 rows are added to the design for each factor), the number you specify *as* the RHS of an *insertion rule* in the **AxialPointGroups** option is the *number of groups* to be inserted. Thus, the default setting will insert one *group* of axial points (i.e., 2**k** rows, where **k** is the number of factors) at the bottom of the design.

In addition, the **CentralPoints** option is set by default to:

CentralPoints->{**Bottom**->**NumberPerFactor**->2}.

This function supports all the options that are described in Appendix 2: Common Options. All of the options that are supported by the **DOEFractional** function are also available in the **DOECentralComposite** function. Thus, you can use the **ConfoundingRules** option to confound some of the factors in the design, thereby reducing the number of runs that would otherwise be required.

Options Shared by DOECentralComposite and DOEFractional:

(See Usage page for DOEFractional)

Options Unique to DOECentralComposite:

(None)

Example:

```
cencompxdsn = DOECentralComposite[<|"X1" -> {0.7, 1.7, 2.7}, "X2" -> {40, 50, 60},
"X3" -> {1.8, 2.3, 2.8}|>, AxialPointGroups -> {DesignAlphaSpherical, {DesignInsertAtTop -> 1}}]
```

Function: **DOEDesirability**

Usage:

DOEDesirability[*fittedModelList*, *desireFcnList*, *xVars*, *xVarNames*, *yVars*, *yVarNames*] creates a grid of dynamic model fit plots based on the FittedModel objects and desirability functions provided by *fittedModelList* and *desireFcnList*. It is assumed that for each **i**, *fittedModelList*[[**i**]] is the FittedModel object returned by Mathematica's LinearModelFit or NonlinearModelFit function using the values of the response variable given by *yVars*[[**i**]] and the values of the independent variables in *xVars*, where *xVars*[[**j**]] gives the values for the **j**-th independent variable. The lists *xVarNames* and *yVarNames* provide the names of the x and y variables in the the fitted model.

Options Unique to DOEDesirability:

MinMaxForXVars->xMinMax, where xMinMax is a list such that xMinMax[[**j**]] is a list containing the min and max values to be used for the **j**-th independent variable when fitting each response variable. By default, xMinMax is set to the empty list. When xMinMax is set to the empty list, the min and max values used for each independent variable are derived from the xVars list.

FitPlotColor->clr, where clr is an RGBColor. This option allows you to control the color of the curves in the Fit plots displayed in the grid. By default, clr is set to Blue.

Example:

DOEDesirability[fittedModelList, desireFcnListDS, xVars, xVarNames, yVars, yVarNames]

Function: **DOEFactorial**

Usage:

xdsn = DOEFactorial[*factorsAndLevels*] generates a full factorial experimental design for the factors described by *factorsAndLevels*. The input argument *factorsAndLevels* is an association, which connects the name of each factor to its level-values. The function returns xdsn, a 2-part list. xdsn[[1]] is a Dataset object that encapsulates the generated runs. xdsn[[2]] contains metadata.

This function supports all the options that are described in Appendix 2: Common Options.

Options Unique to DOEFactorial:

(None)

Example:

```
fullfactxdsn = DOEFactorial[<|"X1" -> {0.7, 1.9}, "X2" -> {40, 60}, "X3" -> {13.2, 13.9}|>]
```

Function: **DOEFractional**

Usage:

xdsn = DOEFractional[*factorsAndLevels*] generates a fractional factorial experimental design for the factors described by *factorsAndLevels*. The input argument *factorsAndLevels* is an association, which connects the name of each factor to its level-values. Factors are confounded using the rules specified by the `ConfoundingRules` option. The function returns xdsn, a 2-part list. xdsn[[1]] is a Dataset object that encapsulates the generated runs. xdsn[[2]] contains metadata.

This function supports all the options that are described in Appendix 2: Common Options.

Options Shared by DOEFractional and DOECentralComposite:

`ConfoundingRules`->conf, where conf is set to a list of confounding rules. Each confounding rule has the form: ConfVar->{Var1,Var2,…,VarK} where ConfVar is the name of a factor and {Var1,Var2,…,VarK} are the names of the factors with which ConfVar is to be confounded. If conf is the empty list then the **DOEFractional** function computes the same design as the **DOEFactorial** function.

`UseInteractionTables`->flag, where flag is either True or False. If flag is set to True, then level multiplication tables are used to compute the products of interactions for 2x2-, 2x3-, and 3x3- level factor interactions. By default, flag is set to True. If flag is set to False, then level lists for confounded factors are ignored when computing the products of interactions. Instead, the value of a confounded factor in a given run is computed by forming the product of the level values in that run of its interacting factors.

`InteractionTable2x2`->levmtab, where levmtab is a list of rules that defines the result when level L_i of factor F1 and level L_j of factor F2 (where both factors have two levels) are multiplied due to factor interaction. By default, this option is set as follows:

```
InteractionTable2x2->
    {
        {1,1}->1,
        {1,2}->1,
        {2,1}->2,
        {2,2}->2
    }
```

`InteractionTable2x3`->levmtab, where levmtab is a list of rules that defines the result when level L_i of factor F1 and level L_j of factor F2 (where the first factor has two levels and the second factor has 3 levels) are multiplied due to factor interaction. By default, this option is set as follows:

```
InteractionTable2x3->
    {
        {1,1}->1,
        {1,2}->2,
        {1,3}->3,
        {2,1}->2,
        {2,2}->3,
        {2,3}->1
    }
```

When the first factor has 3 levels and the second has 2, the LHS of each rule in the **`InteractionTable2x3`** is reversed. Thus, if the default **`InteractionTable2x3`** is being used, then the following table is applied:

```
    {
        {1,1}->1,
        {2,1}->2,
        {3,1}->3,
        {1,2}->2,
```

```
        {2,2}->3,
        {3,2}->1
    }
```

InteractionTable3x3-> levmtab, where levmtab is a list of rules that defines the result when level L_i of factor F1 and level L_j of factor F2 (where both factors have three levels) are multiplied due to factor interaction. By default, this option is set as follows:

```
InteractionTable3x3->
    {
        {1,1}->1,
        {1,2}->1,
        {1,3}->2,
        {2,1}->1,
        {2,2}->2,
        {2,3}->3,
        {3,1}->2,
        {3,2}->3,
        {3,3}->3
    }
```

Options Unique to DOEFractional:

(None)

Example :

```
fractfactxdsn = DOEFractional[<|"X1" -> {0.7, 1.9, 3.1}, "X2" -> {40, 50, 60},
"X3" -> {1.8, 2.3, 2.8}|>, ConfoundingRules -> {"X3" -> {"X1", "X2"}}]
```

Function: **DOEMixture**

Usage:

xdsn = DOEMixture[*factorList*] generates a mixture experimental design for the factors in *factorList,* which is a list of factor names. The function returns xdsn, a 2-part list. xdsn[[1]] is a Dataset object that encapsulates the generated runs. xdsn[[2]] contains metadata.

This function supports all the options that are described in Appendix 2: Common Options.

Options Unique to DOEMixture:

MixtureType->mixtype, where mixtype is set one of the keywords in **{SimplexLattice, SimplexCentroid}**. By default, mixtype is set to **SimplexLattice**.

MixtureLevelsPerFactor->nlev, where nlev is either the keyword **MixtureNumberFactors** (i.e., Length[factorList]) or else an integer >= 2. By default, nlev is set to **MixtureNumberFactors**.

PatternIsBarChart->flag, where flag is either True or False. If flag is set to True, then the Pattern column displays a bar chart for each row of the design to represent the pattern for that row of the design. Otherwise, the pattern for the row is a list of fractions. By default, flag is set to True. This option is ignored if the **PatternColumn** option is set to False.

Example:

```
mixtxdsn = DOEMixture[{"X1","X2","X3","X4"}]
```

Function: **DOEOptimal**

Usage:

x**dsn = DOEOptimal**[*xdsnIn*, *nRuns*] generates an optimal experimental design with *nRuns* runs based on the XDesign object *xdsnIn*. It is assumed that the Dataset component of *xdsnIn* contains main effects only (no computed variables or interactions). The function returns xdsn, a 2-part list. xdsn[[1]] is a Dataset object that encapsulates the generated runs. xdsn[[2]] contains metadata.

This function supports all the options that are described in Appendix 2: Common Options.

Options Shared by DOEOptimal and DOEOrthogonal:

NumberIterations->nIter, where nIter is an integer >= 1. By default, nIter is set to 10. In the case of DOEOptimal, this option is ignored if the **OptimalityMethod** option is set to **Exhaustive**.

Options Unique to DOEOptimal:

OptimalityCriterion->crit, where crit is set one of the keywords in **{OptimalA, OptimalD, OptimalG, OptimalT}.** By default, crit is set to **OptimalD**.

OptimalityMethod->methd, where methd is set one of the keywords in **{CoordinateExchange, RowExchange, Exhaustive}.** By default, methd is set to **RowExchange**.
StartingIndices->startInd, where startInd is a list containing nRuns distinct integers between 1 and N, where N is the number of runs in the Dataset component of the XDesign object xdsnIn. By default, startInd is set to the empty list.

ShowBestCandidates->flag, where flag is either True or False. If flag is set to True, then the list of best candidates discovered in the course of searching for an optimal design will be printed to your notebook. By default, flag is set to False.

Example:

Evaluate the **DOEFactorial** function to create a full factorial design which will be used as input to the **DOEOptimal** function:

```
factorsAndLevels = <|"X1" -> {0.7, 1.9, 3.1}, "X2" -> {40, 50, 60}, "X3" -> {1.8, 2.3, 2.8},
"X4" -> {100, 150}|>;

fullfactxdsn = DOEFactorial[factorsAndLevels, PatternColumn->False,RandomizeRuns->False];
```

Evaluate the **DOEOptimal** function to create a 10-run D-Optimal design based on fullfactxdsn:

```
optimalxdsn = DOEOptimal[fullfactxdsn, 10, OptimalityCriterion->OptimalD]
```

Function: **DOEOrthogonal**

Usage:

x**dsn = DOEOrthogonal**[*factorsAndLevels*] generates a balanced, orthogonal experimental design for the factors described by *factorsAndLevels*. The input argument *factorsAndLevels* is an association, which connects the name of each factor to its level-values. The function returns xdsn, a 2-part list. xdsn[[1]] is a Dataset object that encapsulates the generated runs.
xdsn[[2]] contains metadata.

Each factor must have either 2 or 3 levels.

This function supports all the options that are described in Appendix 2: Common Options.

Options Shared by DOEOrthogonal and DOEOptimal:
(See Usage page for DOEOptimal)

Options Unique to DOEOrthogonal:
(None)

Example:

```
orthogxdsn = DOEOrthogonal[<|"X1" -> {0.7, 1.9}, "X2" -> {40, 50},"X3" -> {1.8, 2.8},
"X4" -> {0.5, 1.5}|>];
```

Function: **DOEToolkitVersion**

Usage:

DOEToolkitVersion[] displays the version of the DOE Toolkit that is loaded in your notebook.

Options

(None)

Example:

DOEToolkitVersion[]

Appendix 2: Common Options for DOE Toolkit Functions

With a few exceptions (as noted), the following options are supported by every function in the DOE Toolkit:

AxialPointGroups->ap, where ap is a list of the form: {**α**, aprules}. You use this option to control the number and locations of axial points that are included in the design. **α** is either a number > 1 or one of the keywords in {**DesignAlphaRotatable, DesignAlphaOrthogonal, DesignAlphaSpherical**}. aprules is a list of rules. For each rule in aprules, the LHS of the rule indicates the insertion location. The LHS of the rule can be one of the keywords {**DesignInsertAtTop, DesignInsertAtBottom, DesignInsertInMiddle**} or else a positive integer (indicating a row number). The RHS of the rule indicates the number of axial point *groups* to be inserted (each group consists of 2**k** rows, where **k** is the number of factors) to insert at the insertion location and is a positive integer. By default, ap is set to the empty list.
Example: **AxialPointGroups**->{**DesignAlphaSpherical,{DesignInsertAtBottom->1**}}

AxialPointsAboveCentralPoints-> flag, where flag is either True or False. If flag is set to true, then when both axial point groups and central points are added to the design, the axial points are inserted above the central points. By default, this option is set to True. This option is ignored if the **CentralPoints** and **AxialPointGroups** options are both set to the empty list.
DataRepresentation->rep, where rep is either a keyword from {**DataAreLevelListValues, DataAreLevelListIndices**} or a list H of symbol-lists. You use this option to control the way the data is displayed in the design. By default, rep is set to **DataAreLevelListValues**.
Example: **DataRepresentation->{{"α","β"},{"ª","©","§"}**

CentralPoints->cp, where cp is a list of rules. You use this option to control the number and locations of center points that are included in the design. Each member of cp is a rule; the LHS of the rule indicates the insertion location. The LHS of the rule can be one of the keywords {**DesignInsertAtTop, DesignInsertAtBottom, DesignInsertInMiddle**} or else a positive integer (indicating a row number). The RHS of the rule indicates the number of central points to insert at the insertion location and is either a positive integer or a rule of the form: **NumberPerFactor->k**, where **k** is a positive integer. By default, cp is set to the empty list.
Example: **CentralPoints**->{**DesignInsertAtTop->NumberPerFactor->1**}

DatasetOptions->dsopts, where dsopts is set to a list of Dataset options. The options in dsopts will be applied to the Dataset component returned by the DOE Toolkit function. By default, dsopts is set to the empty list.
Example: **DatasetOptions**->{**HeaderBackground**->{**LightGreen,Cyan**},
HeaderStyle->Bold}

DesignBlockColors->{c1,c2}, where c1 and c2 are colors. The colors c1 and c2 are used to alternately color the blocks in the design. By default, this option is set to {**LightBlue, White**}. This option is ignored if the **DesignBlocking** option is set to the empty list or if the **DesignBlocksAreRows** option is set to False.

DesignBlocking->blk, where blk.is set to a list of rules of the form:
BFactor->BLevelList. In each rule, BFactor is the name to be assigned to the blocking factor and BLevelList is the list of levels associated with BFactor. By default, blk is set to the empty list (i.e., no blocking is included in the design).
Example: **DesignBlocking**->{"Plot"->{1,2}}

DesignBlocksAreRows->flag, where flag is either True or False. If flag is set to True, then the runs within each block are provided as rows in the Dataset. Otherwise, blocks are provided as columns, one for each combination of factor levels. By default, this option is set to True. This option is ignored if the **DesignBlocking** option is set to the empty list.

ExtraColumns->colList, where colList is a list of names. For each name in colList, a column will be added to the Dataset component of the 2-part list returned by the DOE Toolkit function. By default, colList is set to the empty list. If a particular member of colList is a 2-part list, then the first part will be used as the name of the column and the second part will be used as the initial value in the column for each row. Otherwise, the initial value in the column for each row will be set to Missing.
Example: **ExtraColumns**->{"Y1",{"Y2",0}}

MyMetadata->mdata, where mdata is set to a list of rules supplied by the user. If mdata is set to a non-empty list, mdata will be included in the metadata component of the 2-part list returned by the DOE Toolkit. By default, mdata is set to the empty list.
Example: **MyMetadata**->{**Description**->"Generated "
<>DateString[{"Month","/","Day","/","Year"}]}

NumberReplicates->nr, where nr is a non-negative integer. You use this option to control the number of times each run is replicated in the design. By default, nr is set to 0.

PatternColumn->flag, where flag is either True or False. If flag is set to True, then a column named "Pattern" is included in the Dataset component of the 2-part list returned by the DOE Toolkit function. The "Pattern" column contains expressions that represent the levels of each factor that are present in the generated runs. A pattern for a given run is either a string composed of symbols from the list {-, +, 0} or a list of level indices. By default, flag is set to True.

PatternRepresentation->rep, where rep is a keyword from {**PatternsAreLevelListIndices, PatternsAreMinusZeroPlus**} or a list H of symbol-lists. You use this option to control the way the Pattern column is displayed in the design. By default, rep is set to **PatternsAreLevelListIndices**. If rep is set to **PatternsAreMinusZeroPlus** and some factor has only 1 level, the level is represented by the "*" character in the pattern. If some factor has more than 3 levels, then its levels are represented by indices in the pattern. If rep is set to a list H, then the same rules apply to H as described above for the **DataRepresentation** option. This option is ignored if the **PatternColumn** option is set to False.
Example: **PatternRepresentation**->{{**Red,Green**},{**Orange,Blue,Purple**}}

PrintLog->flag, where flag is either True or False. If flag is set to True, then a log that describes the process that was followed to find the design will be printed to your notebook. By default, flag is set to False.

RandomizeRuns->flag, where flag is either True or False. If flag is set to True, then the runs in the design are randomly ordered. By default, flag is set to True. If the **DesignBlocking** option is specified and the **DesignBlocksAreRows** option is set to True, then the runs within each block are randomly ordered.

RandomNumberSeed->rSeed, where rSeed is either the keyword **BasedOnCurrentTime** or else an integer or a string. By default, rSeed is set to **BasedOnCurrentTime**. When the **RandomizeRuns** option is set to True, the **RandomNumberSeed** option allows you to control whether the randomization performed in creating a design is repeatable. If you evaluate a particular DOE Toolkit function repeatedly with the **RandomNumberSeed** option set to the same value of rSeed each time (where rSeed is set to some fixed integer or string), then the same random ordering of the generated runs will be produced each time. If you don't explicitly set the **RandomNumberSeed** option (in which case, it will be set to its default value of **BasedOnCurrentTime**), then each time you evaluate a particular DOE Toolkit function, a different random ordering of the generated runs will be produced.

RowNumbers->flag, where flag is either True or False. If flag is set to True, then the runs are numbered in the Dataset component of the 2-part list returned by the DOE Toolkit function. By default, flag is set to True.

Appendix 3: Editing a Dataset

When it comes to Mathematica, there will almost always be more than one way to accomplish whatever it is you are trying to do. In many cases, there will be *several* ways to achieve a desired result. In the examples that follow, we show how to perform the most basic row and column operations on a Dataset object. We do not claim that the methods we use are the best or the most efficient in any sense. We believe that the methods we use are fairly clear and should be easy to adapt to suit your needs.

Row Operations

Suppose we create a full factorial design using the **DOEFactorial** function:

```
factorsAndLevels = <|"X1" -> {0.7, 1.7}, "X2" -> {40, 60}, "X3" -> {1.8, 2.8}|>;
fullfactxdsn = DOEFactorial[factorsAndLevels, RandomizeRuns->False]
```

Let's set the local variable ds1 to the first component of the XDesign returned by the DOEFactorial function:

```
ds1 = fullfactxdsn[[1]]
```

The following examples all use ds1 as shown below:

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

Example 1: Append a row to ds1.

Define a new row of data as an Association:

```
newrow = Association[{"Pattern"->"","X1"->1.8,"X2"->55,"X3"->2.9}]
```

The following is printed to your notebook:

```
<|Pattern → , X1 → 1.8, X2 → 55, X3 → 2.9|>
```

Append the rule: "mykey"->newrow to ds1. Note how we use the Key: "mykey" as the LHS of the rule. Any key would do, so long as it is not currently a key in use in ds1.

```
ds1new = Append[ds1,"mykey"->newrow]
```

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8
mykey		1.8	55	2.9

Fix row numbering. This wipes out all current "row number" keys and renumbers them as 1..N:

```
ds1new =Dataset[AssociationThread[Table[i,{i,1,Length[ds1new]}],
Values[Normal[ds1new]]]]
```

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8
9		1.8	55	2.9

Define a helper function to fix row numbering in the following examples:

```
fixRowNumbering[dsIn_]:=
Dataset[AssociationThread[Table[i,{i,1,Length[dsIn]}],Values[Normal[dsIn]]]]
```

Example 2: Insert a row into the position currently occupied by row 6 in ds1.

Insert the rule: "mykey"->newrow into ds1. We use "mykey" for the key in the LHS of the rule, which we know is not currently in use in ds1. We specify the insertion location as Key[6]. Call our helper function to fix row numbering.

```
ds1new=Insert[ds1," mykey"->newrow,Key[6]];
ds1new=fixRowNumbering[ds1new]
```

ds1 at outset

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after insert row, Key[6]

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
mykey		1.8	55	2.9
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after fix row numbering

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6		1.8	55	2.9
7	212	1.7	40	2.8
8	122	0.7	60	2.8
9	222	1.7	60	2.8

Example 3: Delete row 3 in ds1.

```
ds1new = Dataset[KeyDrop[Normal[ds1],3]];
ds1new=fixRowNumbering[ds1new]
```

ds1 at outset

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after delete row

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after fix row numbering

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	221	1.7	60	1.8
4	112	0.7	40	2.8
5	212	1.7	40	2.8
6	122	0.7	60	2.8
7	222	1.7	60	2.8

Example 4: Replace row 3 in ds1 with a different row of values. Note how we re-use newrow, created in example 1.

```
ds1assoc=Normal[ds1];
ds1assoc[3]=newrow;
ds1new=Dataset[ds1assoc]
```

ds1 at outset

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after replace row 3

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3		1.8	55	2.9
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

Example 5: Modify row 3 in ds1 by changing the value of X1 to 6.4.

```
ds1assoc = Normal[ds1];
row3=ds1assoc[3];
row3["X1"]=6.4;
ds1assoc[3]=row3;
ds1new=Dataset[ds1assoc]
```

ds1 at outset

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after modifying row 3

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	6.4	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

Example 6: Move row 7 in ds1 so that it becomes the 4th row.

Step1: Make a copy of row 7 and then delete it.

```
ds1assoc = Normal[ds1];
row7=ds1assoc[7];
ds1assoc = KeyDrop[ds1assoc,7];
```

Step 2: Insert the rule: "mykey"->row7 into ds1. Specify insertion location as Key[4]. Fix row numbering.

```
ds1assoc = Insert[ds1assoc,"mykey"->row7,Key[4]];
ds1new=fixRowNumbering[ds1assoc]
```

ds1 at outset

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after move row 7 to row 4

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	122	0.7	60	2.8
5	221	1.7	60	1.8
6	112	0.7	40	2.8
7	212	1.7	40	2.8
8	222	1.7	60	2.8

Example 7: Swap rows 2 and 7 in ds1.

```
ds1assoc = Normal[ds1];
row2=ds1assoc[2];
ds1assoc[2]=ds1assoc[7];
ds1assoc[7]=row2;
d1new = Dataset[ds1assoc]
```

ds1 at outset

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	211	1.7	40	1.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	122	0.7	60	2.8
8	222	1.7	60	2.8

after swap rows 2 and 7

	Pattern	X1	X2	X3
1	111	0.7	40	1.8
2	122	0.7	60	2.8
3	121	0.7	60	1.8
4	221	1.7	60	1.8
5	112	0.7	40	2.8
6	212	1.7	40	2.8
7	211	1.7	40	1.8
8	222	1.7	60	2.8

Column Operations

The “power tools” we recommend you use to perform column-oriented operations are the **Map** and **MapIndexed** functions.

In the following examples, we use the Map (or MapIndexed) function to perform column operations on our Dataset object ds1. We provide what is known as a *pure* (aka *anonymous*) function – call it *E* – to the Map (or MapIndexed) function, and it applies *E* to each member of the list to which Map (or MapIndexed) is being applied.

If you unfamiliar with pure functions and would like to learn more about them, we suggest you take a look at: https://reference.wolfram.com/language/ref/Function.html

Example 1: Use the **Map** function to add a column "A" of zeroes to ds1.

```
ds1new=Map[Append[#,"A"->0]&,ds1]
```

	Pattern	X1	X2	X3	A
1	111	0.7	40	1.8	0
2	211	1.7	40	1.8	0
3	121	0.7	60	1.8	0
4	221	1.7	60	1.8	0
5	112	0.7	40	2.8	0
6	212	1.7	40	2.8	0
7	122	0.7	60	2.8	0
8	222	1.7	60	2.8	0

When you use the **MapIndexed** function, one of the input parameters to our pure function *E* is the index of the row to which *E* is being applied. We obtain the index via the "#2" part specification which is provided by MapIndexed. (MapIndexed also provides a "#1" part specification, which essentially picks out the members of the list to which MapIndexed is being applied.)

If you are unfamiliar with the MapIndexed function, consult: https://reference.wolfram.com/language/ref/MapIndexed.html

When you apply MapIndexed to an Association (which is the case in this situation, since the Dataset object ds1 is an Association), **#2** picks out the list: `{Key[i]}`. Meanwhile, `First[#2]` picks out the expression: `Key[i]`. Finally, `First[First[#2]]` gives you a numeric index **i**. Typically, **i** is the index we are after. That is why in the examples below, we use `First[First[#2]]` to get index **i**.

Example 2: Use the **MapIndexed** function to add a column "A" to ds1 where the value for row **i** in column A is the row's index. Note how we have used the expression **`First[First[#2]]`** as described above to get the row's index.

```
ds1new=MapIndexed[Append[#1,"A"->First[First[#2]]]&,ds1]
```

	Pattern	X1	X2	X3	A
1	111	0.7	40	1.8	1
2	211	1.7	40	1.8	2
3	121	0.7	60	1.8	3
4	221	1.7	60	1.8	4
5	112	0.7	40	2.8	5
6	212	1.7	40	2.8	6
7	122	0.7	60	2.8	7
8	222	1.7	60	2.8	8

Example 3: Add a column "A" to ds1 where the value for row **i** in column A is derived from a given vector of values: colAvalues1.

```
colAvalues1={101,241,564,111,789,432,905,838};
ds1new=MapIndexed[Append[#1,"A"->colAvalues1[[First[First[#2]]]]]&,ds1]
```

	Pattern	X1	X2	X3	A
1	111	0.7	40	1.8	101
2	211	1.7	40	1.8	241
3	121	0.7	60	1.8	564
4	221	1.7	60	1.8	111
5	112	0.7	40	2.8	789
6	212	1.7	40	2.8	432
7	122	0.7	60	2.8	905
8	222	1.7	60	2.8	838

Example 4: Add a column "A" to ds1 where the value for row **i** in column A is the sum of the values in row **i** of columns X1 and X2.

```
ds1new=Dataset[Map[Append[#,"A"->#["X1"]+#["X2"]]&,ds1]]
```

	Pattern	X1	X2	X3	A
1	111	0.7	40	1.8	40.7
2	211	1.7	40	1.8	41.7
3	121	0.7	60	1.8	60.7
4	221	1.7	60	1.8	61.7
5	112	0.7	40	2.8	40.7
6	212	1.7	40	2.8	41.7
7	122	0.7	60	2.8	60.7
8	222	1.7	60	2.8	61.7

Example 5: Insert a column named "Intercept" (whose value is always 1) before column "X1" in ds1. (The `Key["X1"]` clause indicates that the insertion should be made before column X1.)

```
ds1new=Map[Insert[#1,"Intercept"->1,Key["X1"]]&,ds1]
```

	Pattern	Intercept	X1	X2	X3
1	111	1	0.7	40	1.8
2	211	1	1.7	40	1.8
3	121	1	0.7	60	1.8
4	221	1	1.7	60	1.8
5	112	1	0.7	40	2.8
6	212	1	1.7	40	2.8
7	122	1	0.7	60	2.8
8	222	1	1.7	60	2.8

Example 6: Delete a column in ds1 by key (in this example, we delete column 1, the "Pattern" column).

```
ds1new=Map[Delete[#,1]&,ds1]
```

	X1	X2	X3
1	0.7	40	1.8
2	1.7	40	1.8
3	0.7	60	1.8
4	1.7	60	1.8
5	0.7	40	2.8
6	1.7	40	2.8
7	0.7	60	2.8
8	1.7	60	2.8

Example 7: Delete a column in ds1 by name (in this example, we delete column "X2").

```
ds1new=Dataset[Map[KeyDrop[#,"X2"]&,ds1]]
```

	Pattern	X1	X3
1	111	0.7	1.8
2	211	1.7	1.8
3	121	0.7	1.8
4	221	1.7	1.8
5	112	0.7	2.8
6	212	1.7	2.8
7	122	0.7	2.8
8	222	1.7	2.8

Example 8: Modify/Replace an existing column. In this example, we modify column X2 (whose key is 3) by replacing its values with their square roots.

```
ds1new=Dataset[Map[ReplacePart[#,3->Sqrt[#["X2"]]]&,ds1]]
```

	Pattern	X1	X2	X3
1	111	0.7	6.32456	1.8
2	211	1.7	6.32456	1.8
3	121	0.7	7.74597	1.8
4	221	1.7	7.74597	1.8
5	112	0.7	6.32456	2.8
6	212	1.7	6.32456	2.8
7	122	0.7	7.74597	2.8
8	222	1.7	7.74597	2.8

Example 9: Move a column. In this example, we "move" column X2 by making a back-up, deleting it in its current location, and then inserting it in its new location (before the "Pattern" column).

```
colX2 = ds1[All,"X2"]
ds1new1=Dataset[Map[KeyDrop[#,"X2"]&,ds1]]
ds1new2=MapIndexed[Insert[#1,"X2"->colX2[[First[First[#2]]]], Key["Pattern"]]&,ds1new1]
```

	X2	Pattern	X1	X3
1	40	111	0.7	1.8
2	40	211	1.7	1.8
3	60	121	0.7	1.8
4	60	221	1.7	1.8
5	40	112	0.7	2.8
6	40	212	1.7	2.8
7	60	122	0.7	2.8
8	60	222	1.7	2.8

Last but not least, if you prefer to work with lists rather than associations, you can always extract just the data from the Dataset object, free from all the nested associations, as follows:

```
dataOnly=Map[Values,Normal[Values[ds1]]]
```

The following output is produced:

```
{{111, 0.7, 40, 1.8}, {211, 1.7, 40, 1.8}, {121, 0.7, 60, 1.8},
{221, 1.7, 60, 1.8}, {112, 0.7, 40, 2.8}, {212, 1.7, 40, 2.8}, {122, 0.7, 60, 2.8}, {222, 1.7, 60, 2.8}}
```

Appendix 4: Suggested Resources

The DOE Toolkit assumes that you are already familiar with the process of designing an experiment and analyzing its results. If this is not the case, there are many excellent books and websites which can help you get started with designing your experiments.

Books:

A very gentle introduction to DOE: Gunter, B.; Coleman, D. (2014) **A DOE Handbook: A Simple Approach to Basic Statistical Design of Experiments**.

A classic in the field: Box, George, E.P.; Hunter, William G.; Hunter, J. Stuart (2005).**Statistics for Experimenters: Design, Innovation, and Discovery** (2nd ed).

A very accessible book for the general audience: Cox, D.R. & Reid, N. (2000), **The Theory of the Design of Experiments**.

Websites:

The **National Institute of Standards and Technology** (aka NIST) has created a marvelous website which provides an **Engineering Statistics Handbook**. *Chapter 5 (Improve)* of the handbook contains a wealth of information about the different types of experimental designs. It goes into considerable detail, describing the different types of experimental designs, the purposes they serve, and their strengths and limitations:

https://www.itl.nist.gov/div898/handbook/pri/pri.htm

The **Eberly College of Science at PennState** has an online course that provides an introduction to design of experiments. Although the examples it provides are usually created using Minitab, you don't have to be a Minitab user in order to follow the discussion and gain a lot of useful information.

https://online.stat.psu.edu/stat503/

SAS provides a very useful online technical note: *Experimental Design: Efficiency, Coding, and Choice Designs*. Once again, although the coded examples in the note are written in the SAS programming language, the discussion of experimental design is independent of SAS and is very illuminating:

https://support.sas.com/techsup/technote/mr2010c.pdf

Appendix 5: References

[1] Ruth K. Meyer & Christopher J. Nachtsheim (1995) The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs, Technometrics Vol. 37, No. 1 (Feb., 1995), pp. 60-69.

[2] Fedorov, V. V. (1972), "Theory of Optimal Experiments (Review)", Biometrika, cvol. 59, no. 3, 697-698.
Translated and edited by W. J. Studden and E. M. Klimko.

[3] George Derringer & Ronald Suich (1980) Simultaneous Optimization of Several Response Variables, Journal of Quality Technology, 12:4, 214-219, DOI: 10.1080/00224065.1980.11980968

[4] **Engineering Statistics Handbook, National Institute of Standards and Technology** (aka NIST) website: https://www.itl.nist.gov/div898/handbook

Index