Computer Experiments

Designs

Differences between physical and computer experiments

Recall

1. The code is deterministic. There is no random error (measurement error). As a result, *no replication is needed.* Uncertainty is only due to a lack of knowledge about the true nature of the relationship between the inputs and the outputs. The code is a sort of "black box."

2. Because we know the code, we know all the factors that affect the output. Thus, techniques such as randomization and blocking are not needed, because there is no need to control for the effects of factors that affect the response but have not been included among the experimental factors.

Notation

Recall

The code produces r deterministic outputs $y_1(\boldsymbol{x})$, $y_2(\boldsymbol{x})$, …, $y_r(\boldsymbol{x})$ that depend on a set of input variables

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^\top$$

(For purposes of design, I will usually assume $r = 1$)

We assume the input variables are restricted to some subset

$$\mathbf{X} \subset \Re^d$$

We will assume that our design consists of n points

$$\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$$

Some important features designs for computer experiments should have

1. No replication.

(But note. Occasionally the producer of the code modifies it without telling the statistician. Replication can be useful for verifying that the code has not changed.

2. The design should cover the set of possible input values, **X**, reasonably well.

The rationale for this is that we often don't know the nature of the functional relationship between the inputs and the outputs.

Is it very smooth and slowly varying?

Where are interesting features (optima, regions where the gradient is very steep) located? Lacking such knowledge, they are likely to be anywhere in **X**.

Note: Many popular "classical" designs used in response surface methodology, such as central composite designs and fractional factorial designs, do not spread points out evenly over **X**. Instead they tend to place points at the boundary of **X** (corners, midpoints of faces). As a consequence, they may not be suitable for computer experiments, unless we know that a second order response surface will provide a very good approximation to the output.

3. Designs will usually take relatively few observations. Thus, we seek classes of designs that do not require many observations.

Some strategies for selecting a design

For simplicity, I assume that $\mathbf{X}$ is rectangular of the form

$$\mathbf{X} = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$$

In many of the examples, I will make things even simpler by assuming $\mathbf{X}$ is the d-dimensional unit cube

$$\mathbf{X} = [0, 1]^d$$

Sampling based designs

People typically interpret "covering the set of possible input values, **X**, well" as implying that designs should spread points evenly over **X**. From basic sampling theory, two simple strategies are:

1. Select a random sample of n points

$$\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$$

from $\mathbf{X}$. Denote the i-th point as

$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^\top$$

2. Select a stratified sample of n points in $\mathbf{X}$.

3. Select a Latin hypercube sample (LHS) of n points in $\mathbf{X}$.

Constructing a Latin hypercube sample (LHS)

Divide the range of each interval $[a_j, b_j]$ into n subintervals of equal length. Randomly select a value from each of these subintervals. Let x_{ij} be the value selected for sub-interval i ($1 \leq i \leq n$).

Note: It is not uncommon to take the x_{ij} to be the center of the interval rather than a randomly selected point in the interval.

Form the $n \times d$ array

$$
\begin{array}{llll}
x_{11} & x_{12} & \cdots & x_{1d} \\
x_{21} & x_{22} & \cdots & x_{2d} \\
 & & \vdots & \\
x_{n1} & x_{n2} & \cdots & x_{nd}
\end{array}
$$

Randomly permute each column, using independent permutations for each. The n rows of the resulting array define the points that are our Latin hypercube sample.

Roughly speaking, a Latin hypercube sample divides each dimension of $\mathbf{X}$ into n intervals. n points in $\mathbf{X}$ are selected with the property that when projected onto any dimension, exactly one point is in each of the intervals for that dimension.

Example of a 5 point LHS

For simplicity, assume $\mathbf{X} = [0, 1]^d$.

Divide each [0, 1] into the 5 subintervals [0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.0].

For simplicity, lets use the center of each subinterval.

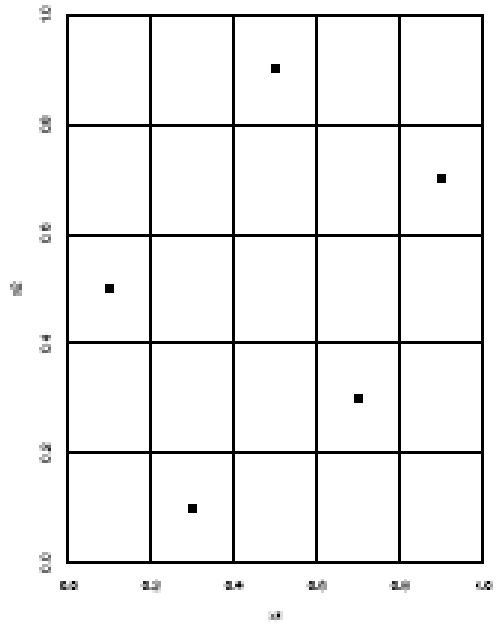
Form the 5×2 array

0.1	0.1
0.3	0.3
0.5	0.5
0.7	0.7
0.9	0.9

Randomly permute each column. One possible result is the array

$$\begin{array}{cc} 0.5 & 0.9 \\ 0.7 & 0.3 \\ 0.3 & 0.1 \\ 0.1 & 0.5 \\ 0.9 & 0.7 \end{array}$$

The resulting 5 point LHS.



$$\Pi = \begin{pmatrix} 3 & 5 \\ 4 & 2 \\ 2 & 1 \\ 1 & 3 \\ 5 & 4 \end{pmatrix}$$

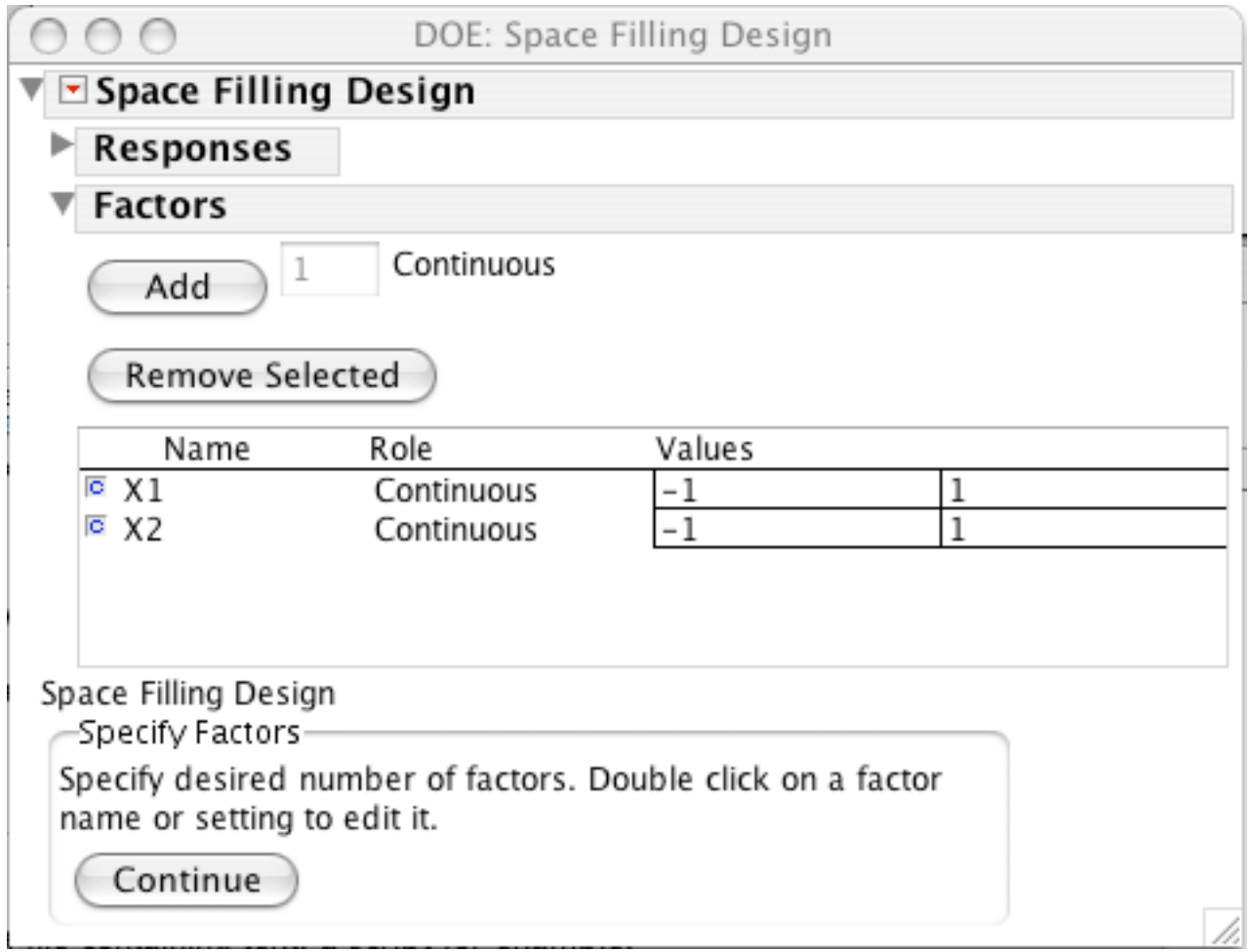
You can generate LHS's using JMP.

1. Under the DOE menu, select Space Filling Design.

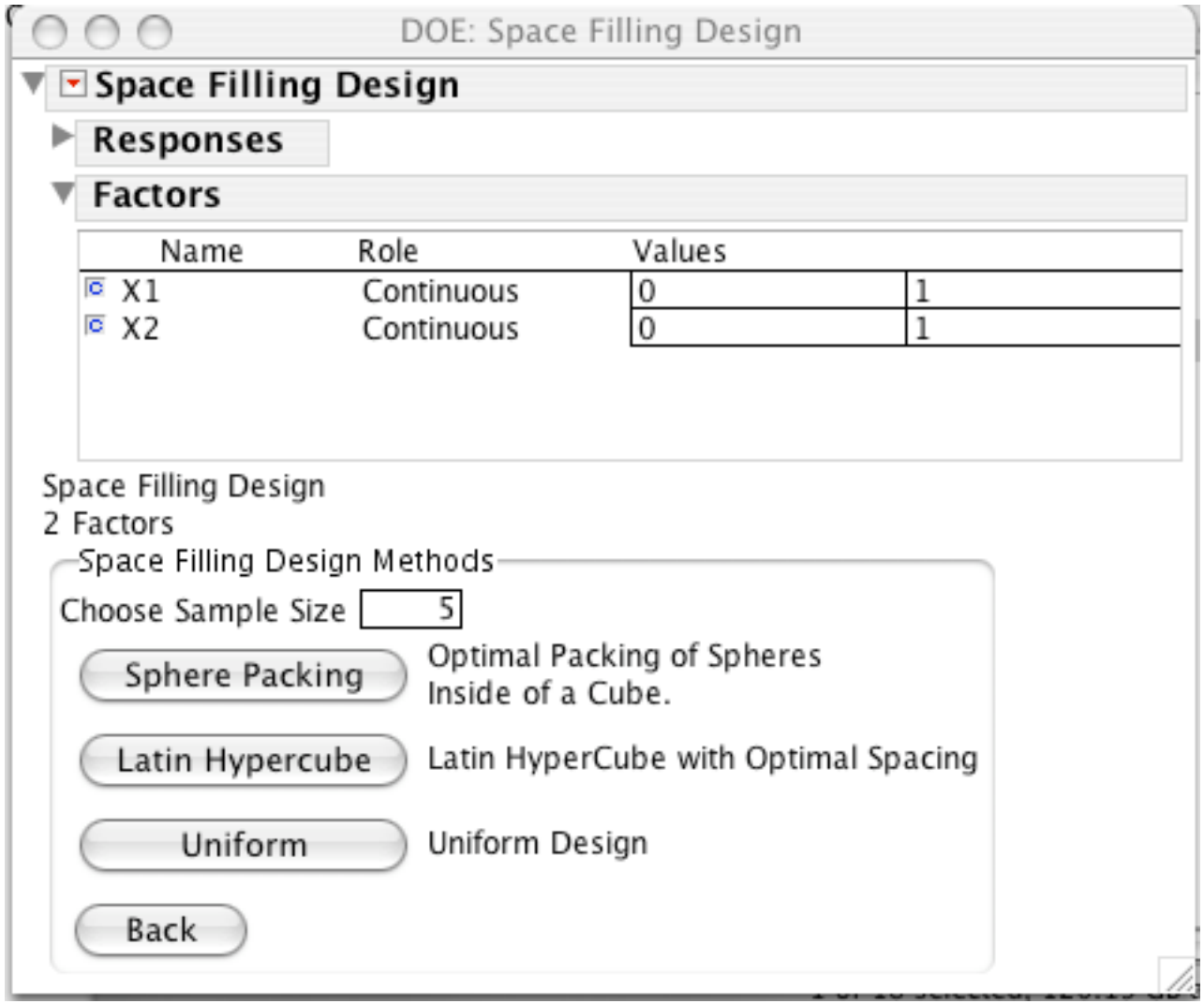
2. Add the appropriate number of factors and specify their range (Values). Click Continue.

3. Select the Sample Size. Click on Latin Hypercube.

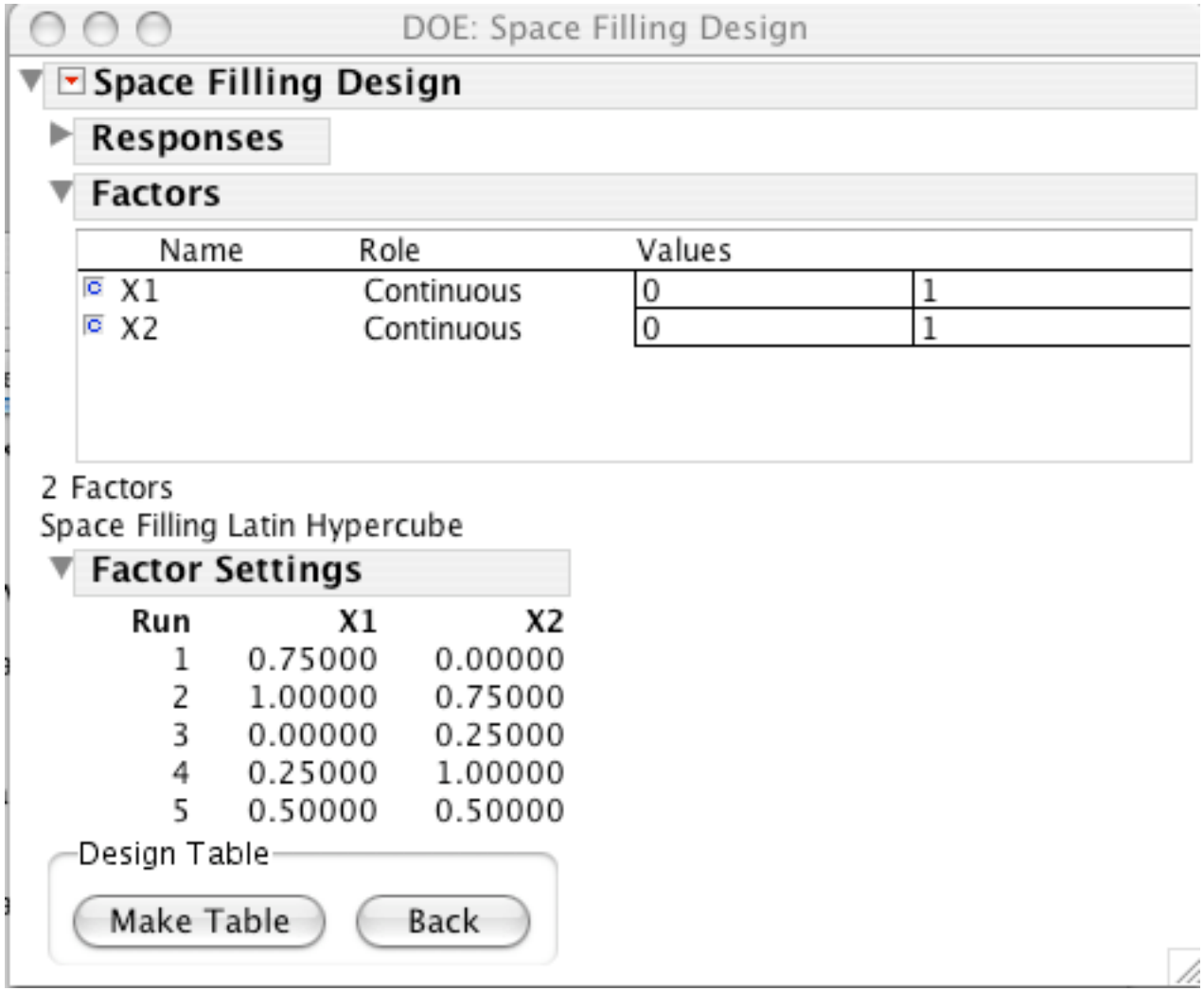
Initial space filling design screen.



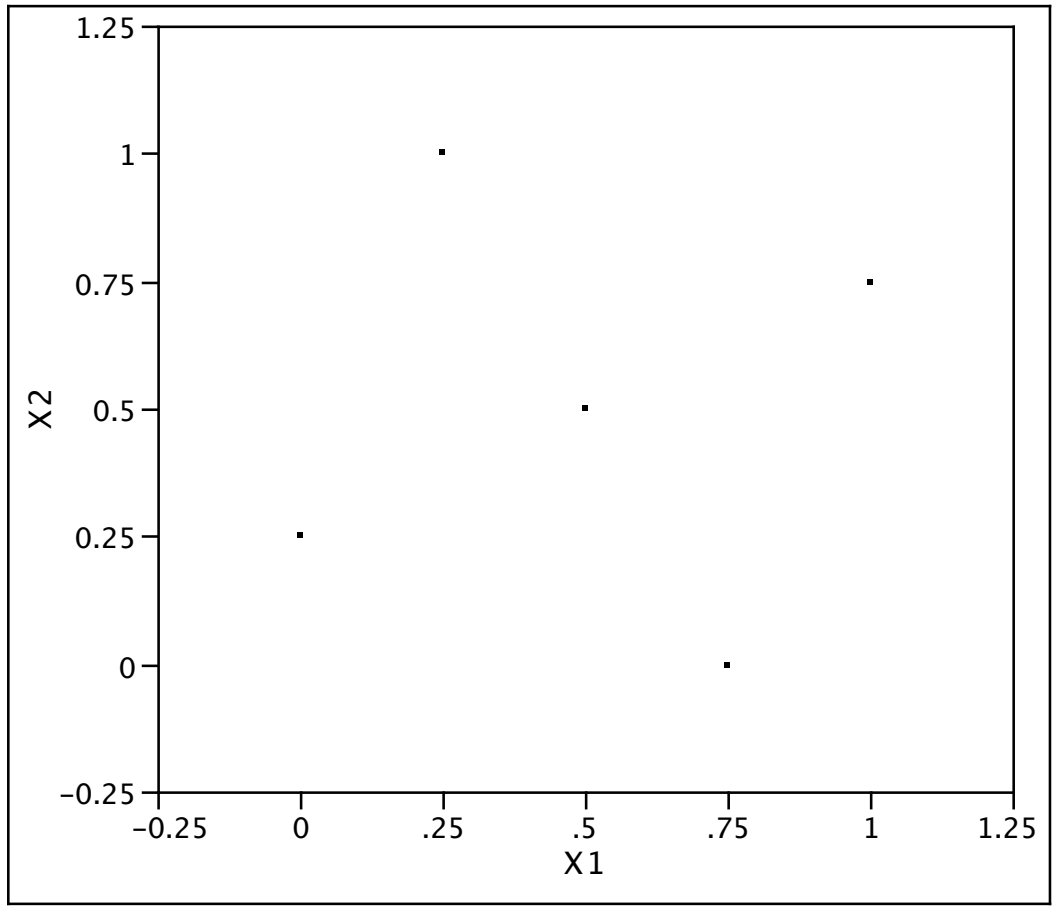
Selecting a space filling design



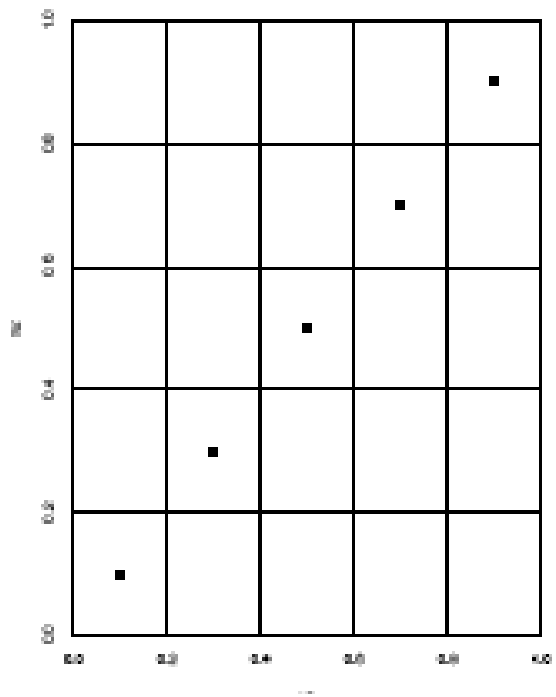
The resulting design



Plot of points



Not all LHS's look uniformly spread out over **X**. Here is a less than ideal 5 point LHS.



Properties of LHS's

McKay, Beckman, and Conover (1979) introduced Latin hypercube sampling and subsequently other authors (for example, Stein 1987 and Owen 1992) have explored their properties. Roughly speaking, suppose we want to find the mean of some known function $G(y(\boldsymbol{x}))$ over $\mathbf{X}$. Then the sample mean of $G(y(\boldsymbol{x}))$ computed from a Latin hypercube sample usually has smaller variance than the sample mean computed from a simple random sample.

Problem: In the computer experiment setting we are considering we are usually not interested in estimating the mean of some $G(y(\boldsymbol{x}))$ over $\mathbf{X}$. More typically we are interested in predicting $y(\boldsymbol{x})$ at previously unobserved inputs.

Some comments

1. When projected onto any dimension, the points in a Latin hypercube sample are spread evenly over that dimension. If $G(y(\boldsymbol{x}))$ is approximately additive, then uniformity in each dimension is intuitively appealing. This, coupled with the fact that they are relatively easy to generate, appears to account for their popularity in practice.

2. There are lots of variations and generalizations of Latin hypercube sampling, including cascading Latin hypercubes, selecting from the class of all Latin hypercubes of size n the one with some additional optimal property, orthogonal arrays (generalize Latin hypercubes to designs with uniform projections onto higher dimensional subspaces).

Distance based designs

1. Minimax or Maximin Designs (see Johnson, Moore, and Ylvisaker (1990) or Koehler and Owen (1996)).

A collection of n points in **X** is a minimax design if it is a collection of *n* points in **X** with the property that the maximum distance between this set of *n* points and all other points in **X** is minimized.

A collection of n points in $\mathbf{X}$ is a maximin design if the minimum distance between any pair of these n points is maximized.

Minimax designs are difficult to generate because distances from all other points in **X** (containing infinitely many points) must be evaluated. Very little in the way of software for generating these designs exists.

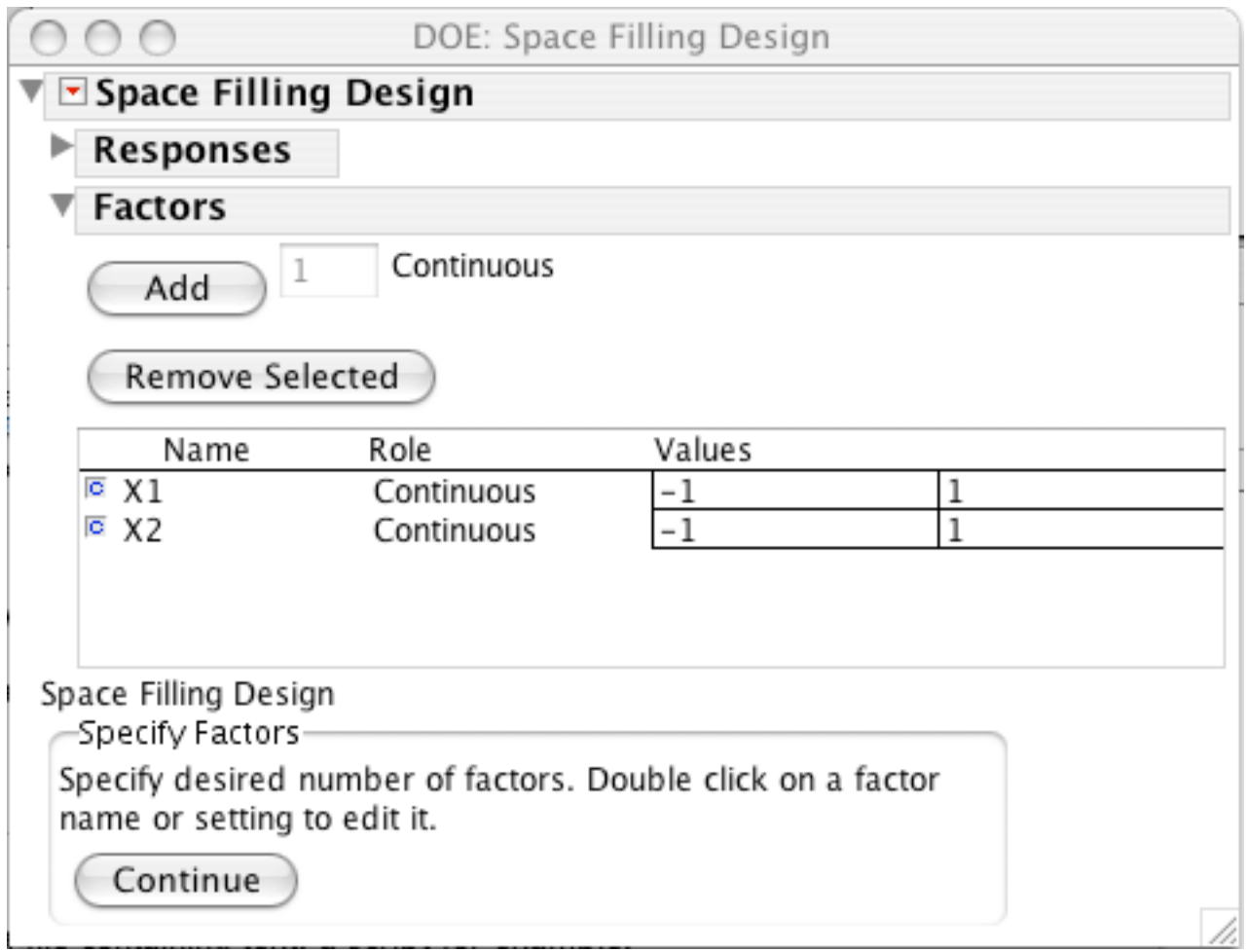
Maximin designs are relatively easy to generate. You can generate these on JMP.

1. Under the DOE menu, select Space Filling Design.

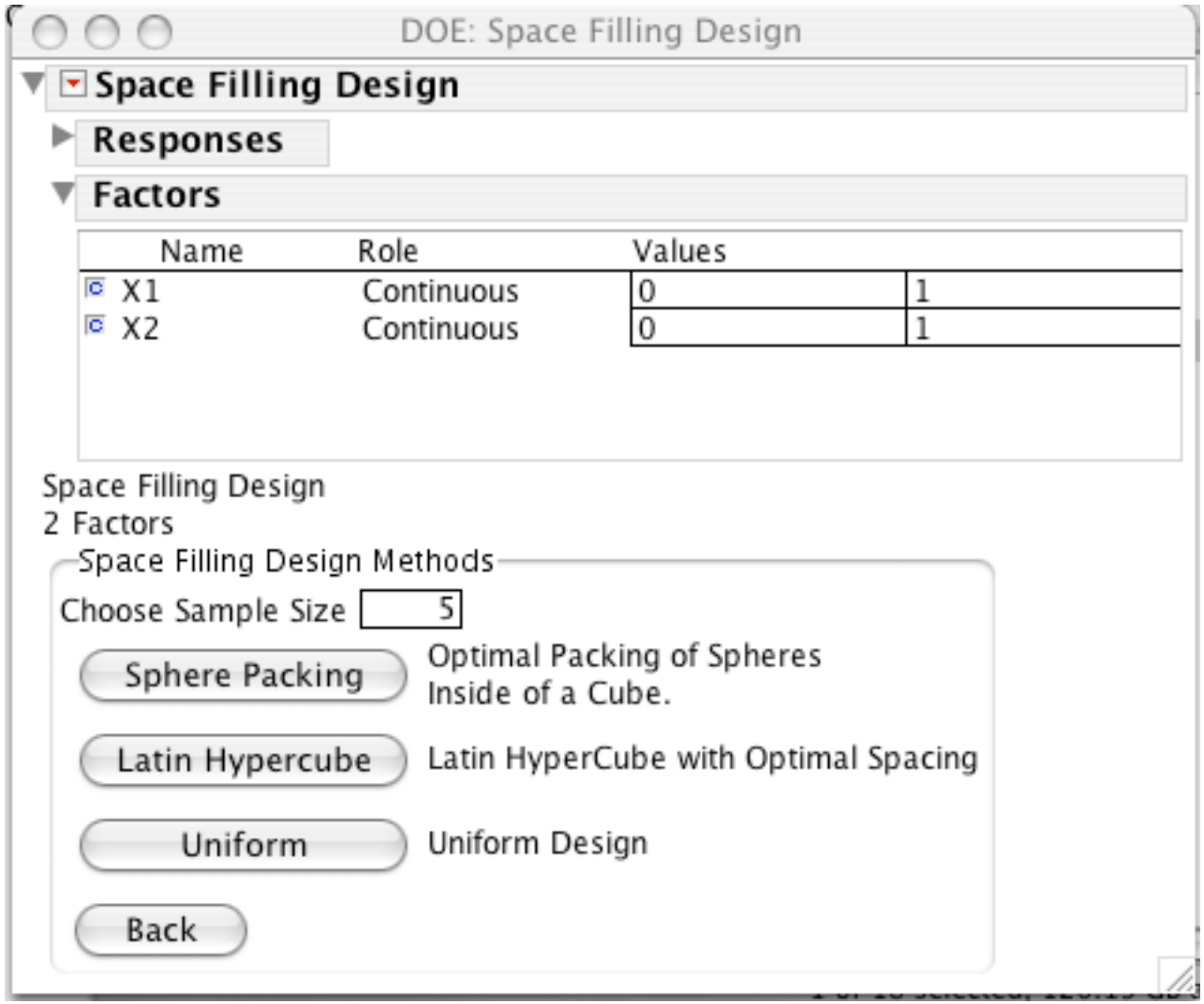
2. Add the appropriate number of factors and specify their range (Values). Click Continue.

3. Select the Sample Size. Click on Sphere Packing.

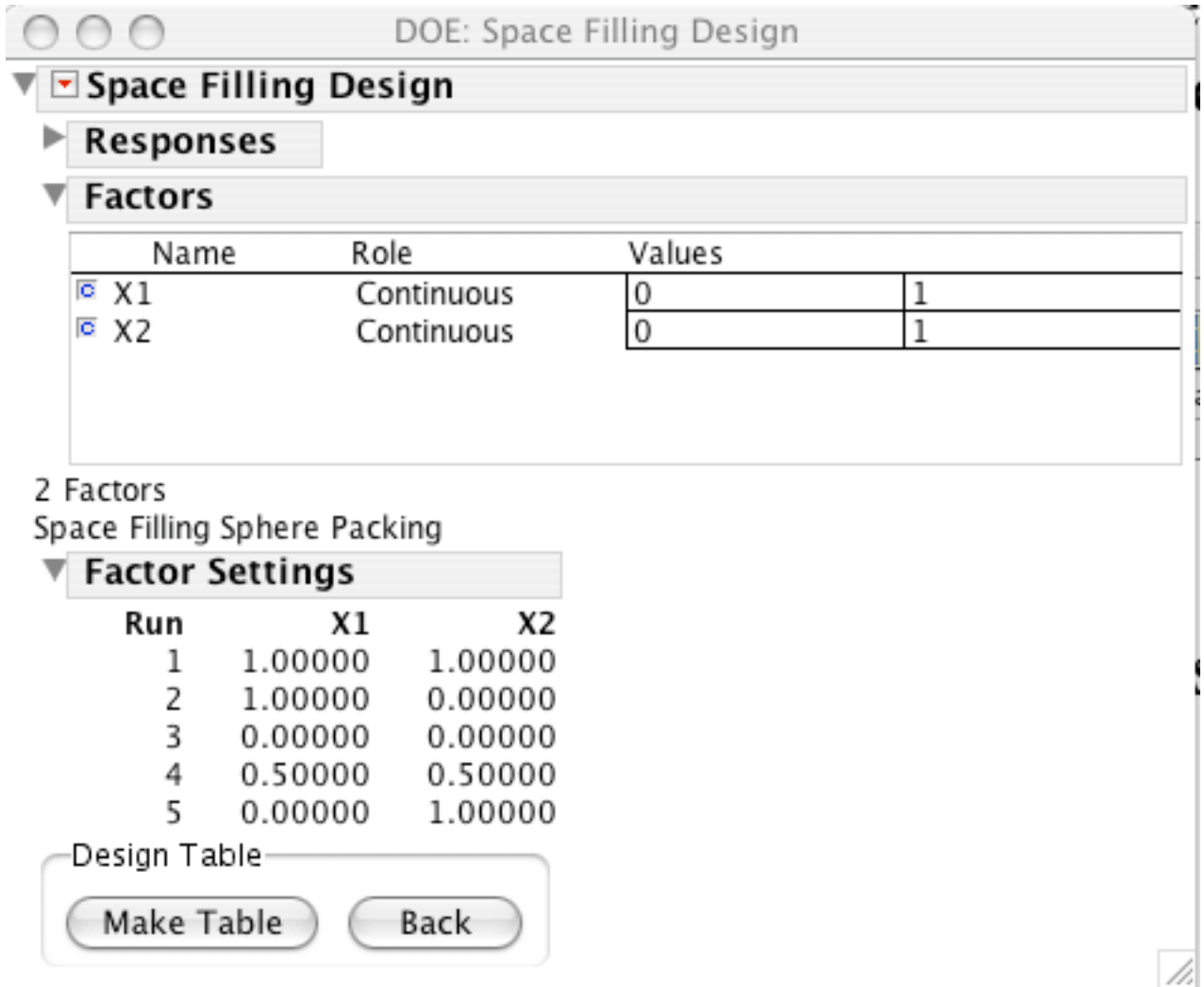
Initial screen



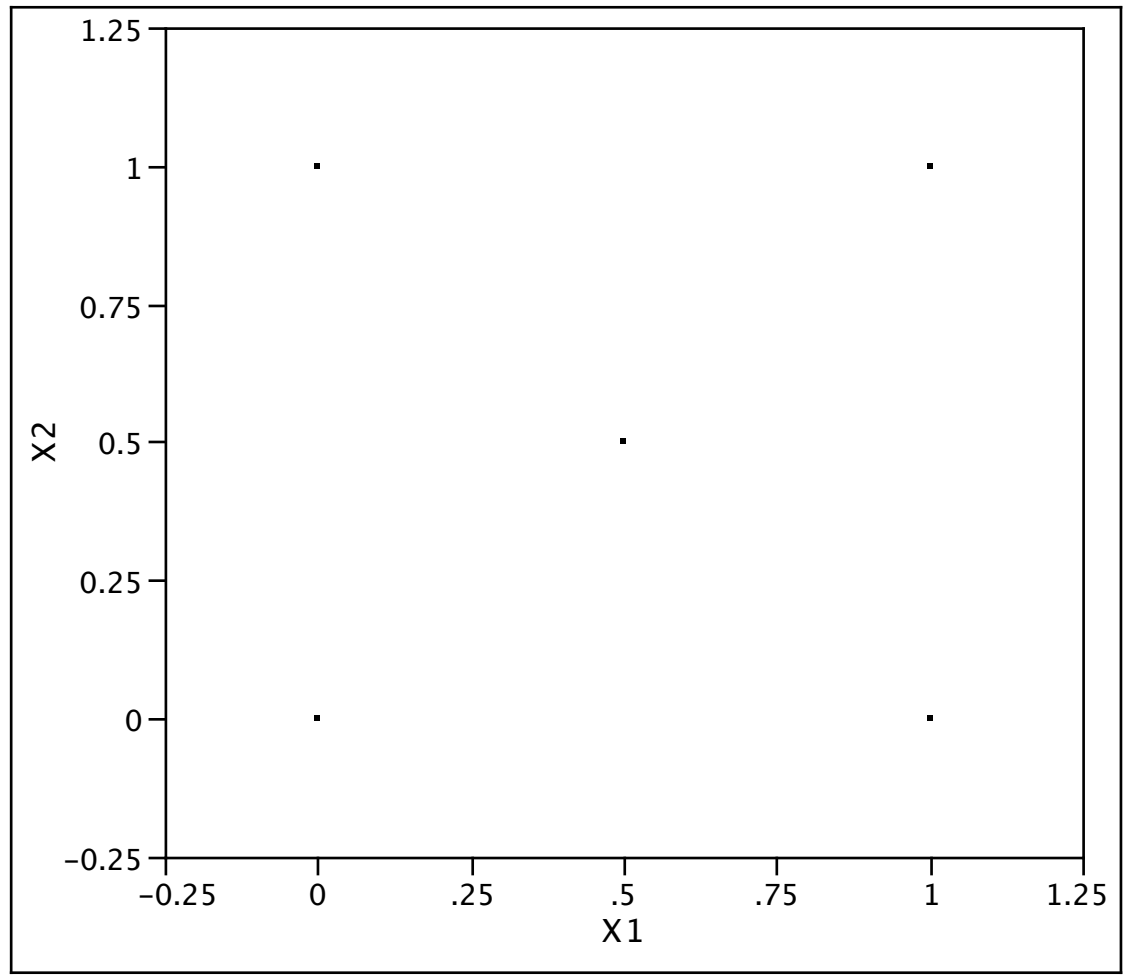
Selecting a 5 point maximin design.



The resulting design



A plot of the points in the design.



Properties of mimimax and maximin designs

Minimax designs might be considered appealing for the following reason. Suppose we fit a model that interpolates the data (the best linear unbiased predictor of a stationary Gaussian process model has this property). Suppose that the discrepancy between our fitted model and the model generated by the computer code increases in a fixed monotonic way as the distance from a point in our design increases. Then minimax designs have a minimax property, namely they minimize the maximum (over $\mathbf{X}$) absolute difference between our fitted model and the model generated by the compute code.

Johnson, Moore, and Ylvisaker (1990) have shown that maximin designs have a sort of D-optimality property under certain conditions. These conditions assume that a certain stationary Gaussian process model is fit to the data and that the prior correlation between points is extremely weak. In fact, D-optimality is in the limit as this correlation goes to 0 (so that the model looks something like the standard linear model with i.i.d. errors).

Note: The LHS's generated by JMP are actually maximin LHS's. By this, I mean that from among all possible LHS's, JMP picks the one that is maximin. By so doing, JMP eliminates "poor" LHS's and guarantees that it generates an LHS that spreads points out reasonably evenly.

Compared to other software that generates LHS's and maximin designs, JMP performs quite well.

2. Uniform designs.

Uniform designs are a collection of points that minimize some measure of discrepancy (departure or distance from a uniform distribution) and are discussed in Fan, Lin, Winker, Zhang (2000).

To be more specific, let

$$D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$$

denote an n point design. Let

$$F_n(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} I[\boldsymbol{x}_i \leq \boldsymbol{x}]$$

where I is the indicator function and the inequality is with respect to componentwise ordering of vectors in d-dimensional Euclidean space.

The L_p discrepancy of D is defined to be

$$\left(\int_{\mathbf{X}} |F_n(\boldsymbol{x}) - F(\boldsymbol{x})|^p \, d\boldsymbol{x} \right)^{\frac{1}{p}}$$

where $F(\boldsymbol{x})$ is the uniform distribution on $\mathbf{X}$.

In the limit as p goes to infinity, this is called the star discrepancy or just the discrepancy of D.

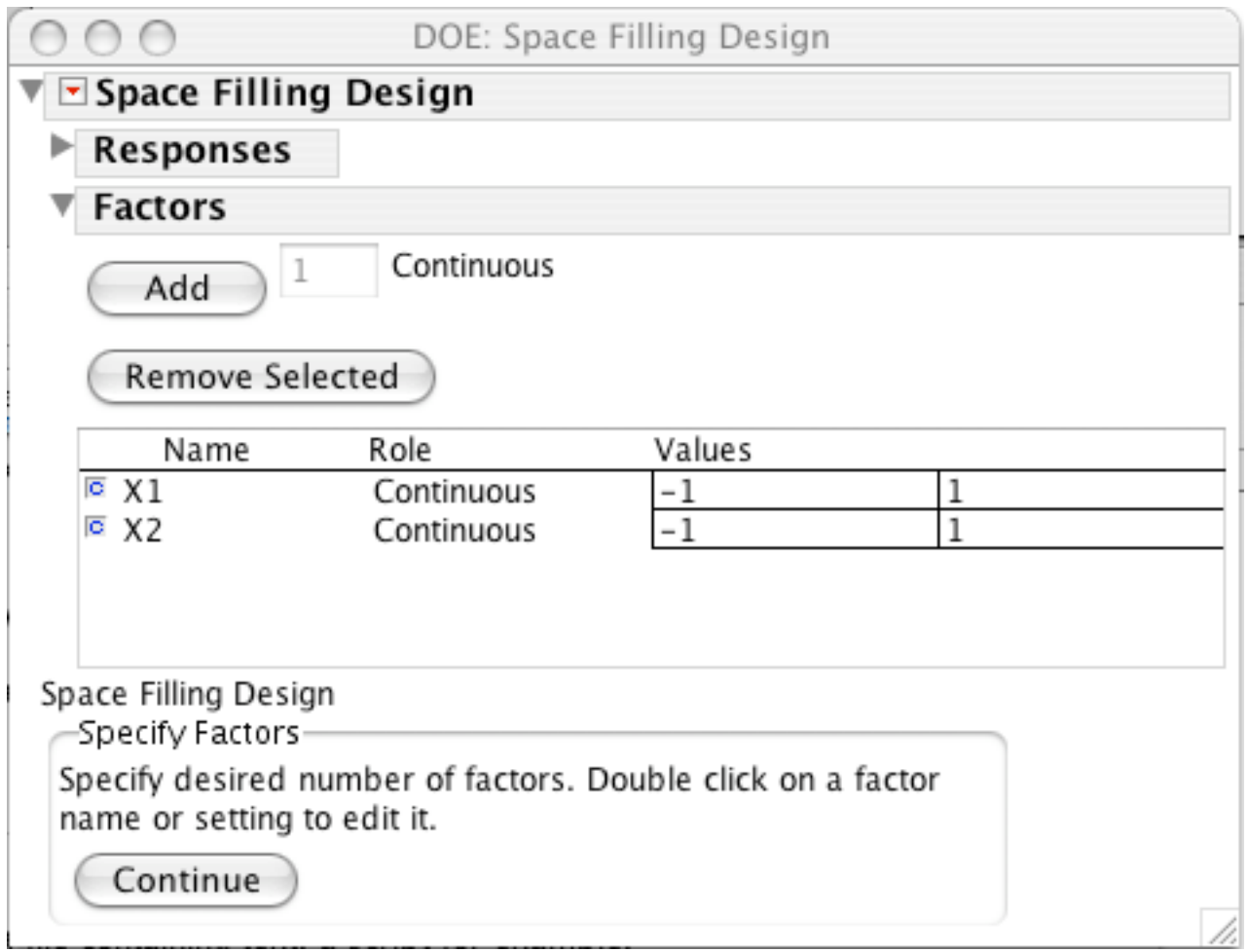
You can generate uniform designs (with respect to the star discrepancy) on JMP.

1. Under the DOE menu, select Space Filling Design.

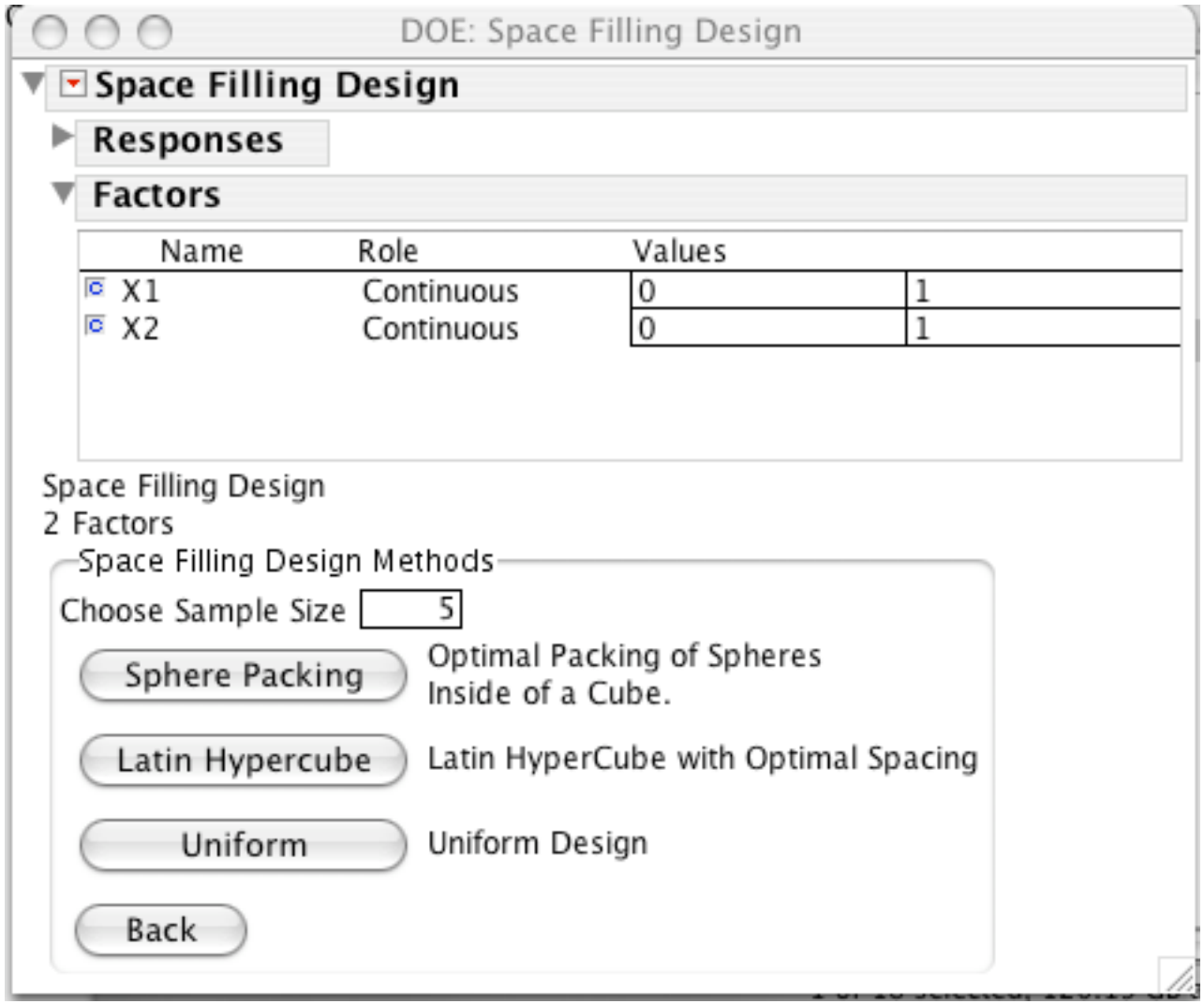
2. Add the appropriate number of factors and specify their range (Values). Click Continue.

3. Select the Sample Size. Click on Uniform.

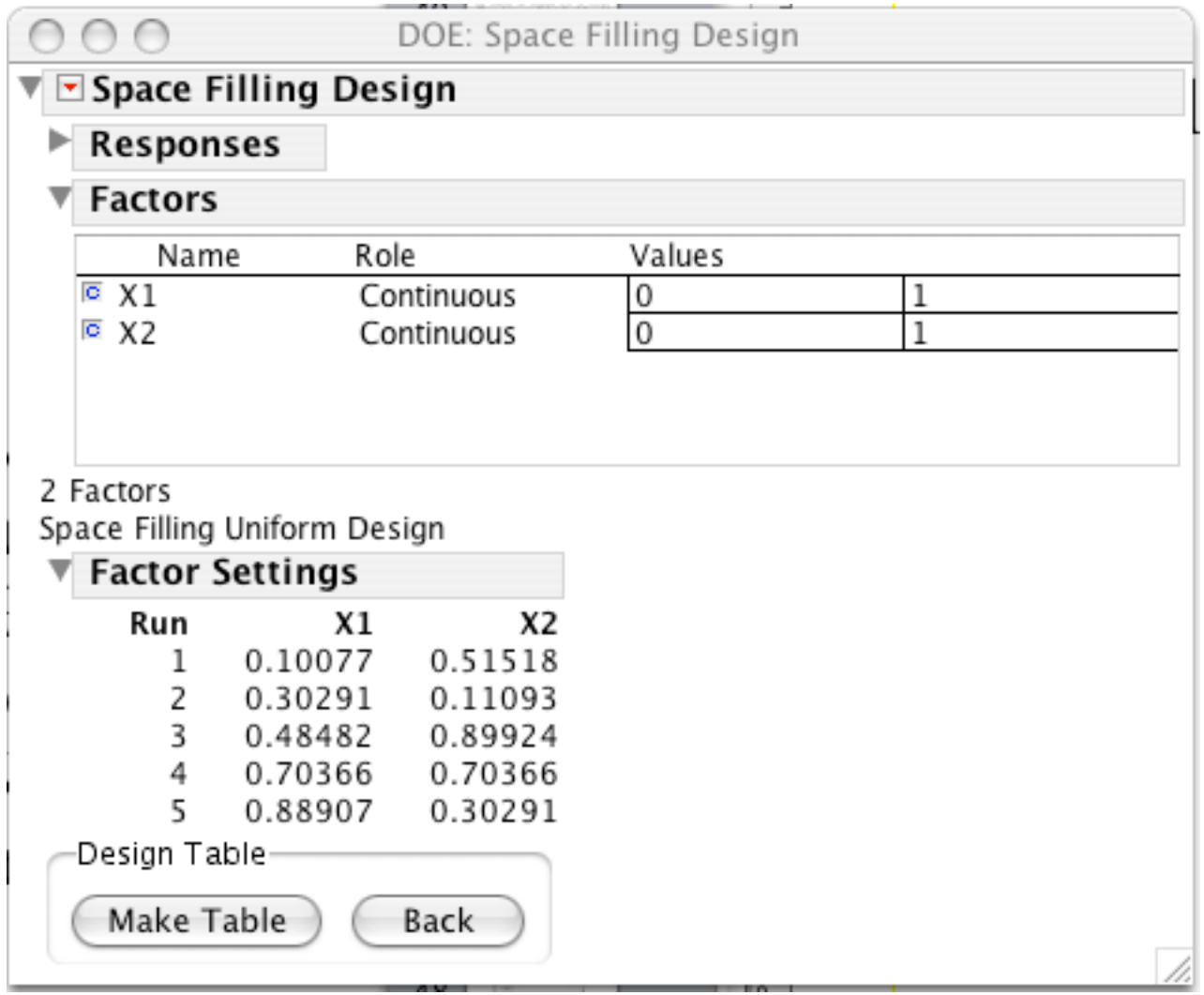
Initial screen



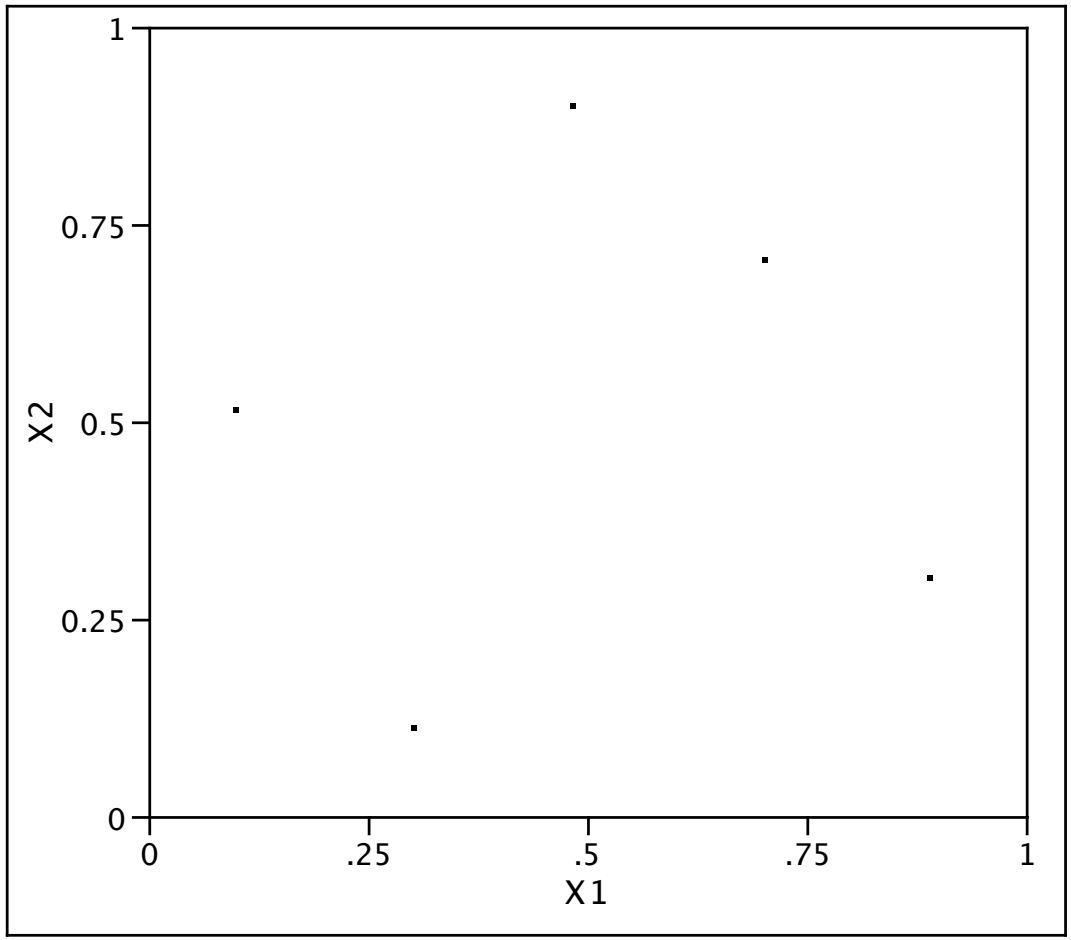
Selecting a 5 point uniform design.



The resulting design



A plot of the points in the design.



Properties of uniform designs

Suppose we want to find the mean of some known function G($y(\boldsymbol{x})$) over $\mathbf{X}$. Than the sample mean of G($y(\boldsymbol{x})$) computed from a uniform design minimizes a certain bound (the Koksma-Hlawka inequality - see Niederreiter 1992 for details) on the absolute error from the true mean.

Problem: In the computer experiment setting we are considering we are not interested in estimating the mean of some $G(y(\boldsymbol{x}))$ over X.

Designs based on formal criteria

A number of "optimality" criteria have been proposed for designs for computer experiments.

Suppose $\hat{y}_D(\boldsymbol{x})$

is our statistical predictor of $y(\boldsymbol{x})$ computed from some design $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$. Let

$$\mathrm{MSE}[\hat{y}_D(\boldsymbol{x})]$$

denote the mean squared error of our predictor.

1. The integrated mean squared error (IMSE) for design *D* is proportional to

$$\int_{\mathbf{X}} \mathrm{MSE}[\hat{y}_D(\boldsymbol{x})] w(\boldsymbol{x}) d\boldsymbol{x}$$

where $w(\boldsymbol{x})$ is some weight function on **X**. A design D^* that minimizes this over the class of all possible *n* point designs on **X** is said to be IMSE-optimal.

Sacks, Schiller, and Welch (1989) discuss these designs.

2. The maximum mean squared error (MMSE) for a design *D* is proportional to

$$\max_{\boldsymbol{x} \in \mathbf{X}} \mathrm{MSE}[\hat{y}_D(\boldsymbol{x})]$$

A design *D** that minimizes MMSE over the class of all possible *n* point designs on **X** is said to be MMSE-optimal.

Sacks and Schiller (1988) discuss these designs.

Problem: For statistical models such as stationary Gaussian process models using the best linear unbiased predictor as our statistical predictor (to be discussed later), one needs to know the correlation parameters to compute these designs. Generally these parameters are unknown and must be estimated from data.

As a consequence, these and other criterion-based designs (for example, D-optimality) are not often used in computer experiments because they depend on unknown parameters.

Designs from Monte Carlo methods

The Monte Carlo method literature includes a variety of designs that are space-filling and useful for Monte carlo integration. Designs include scrambled nets, lattice designs, and sequential designs.

The problem with these designs is that there is little software available to generate these design.

Of these, perhaps the most useful are so-called Sobol sequences. These are space-filling designs with the property that a design with sample size n is obtained from the design of sample size n–1 by adding a point to the design. Most space filling designs do not have this property (for example, a 5 point LHS is not obtained by adding a point to a 4 point LHS).

Code exists for generating these sequences.

1. Apparently R code exists for Sobol sequences - see the Web site

www.maths.lth.se/help/R/.R/library/fOptions/html/D1-LowDiscrepancy.html

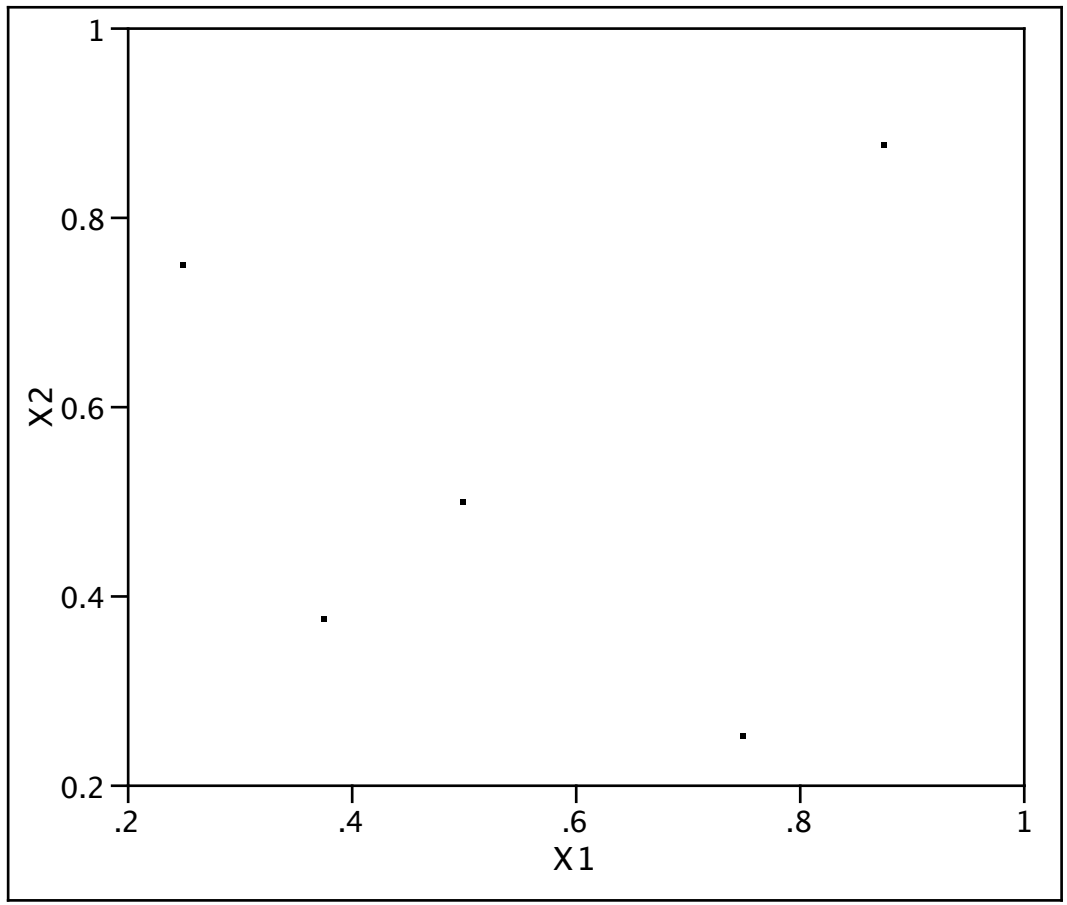
2. There is also C++ code available - see the Web site

www.cosy.sbg.ac.at/~rschuer/hintlib/

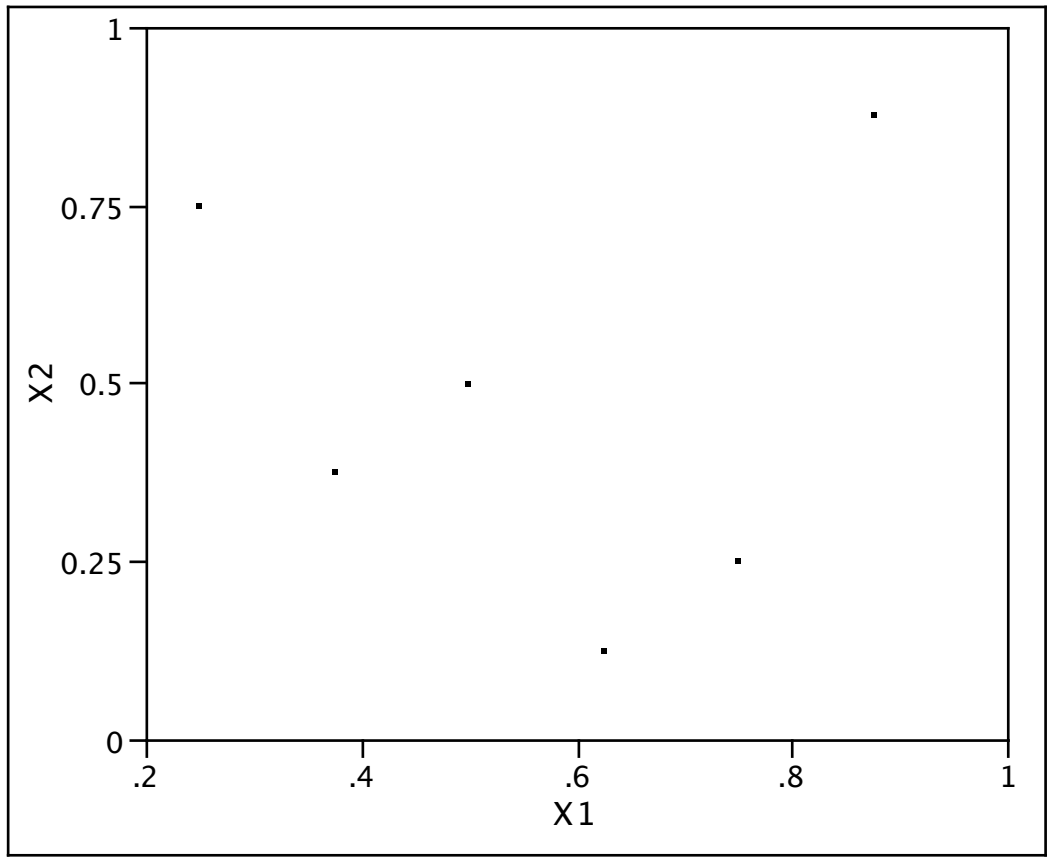
3. I also found a link to software at

/www.broda.co.uk/software.htm

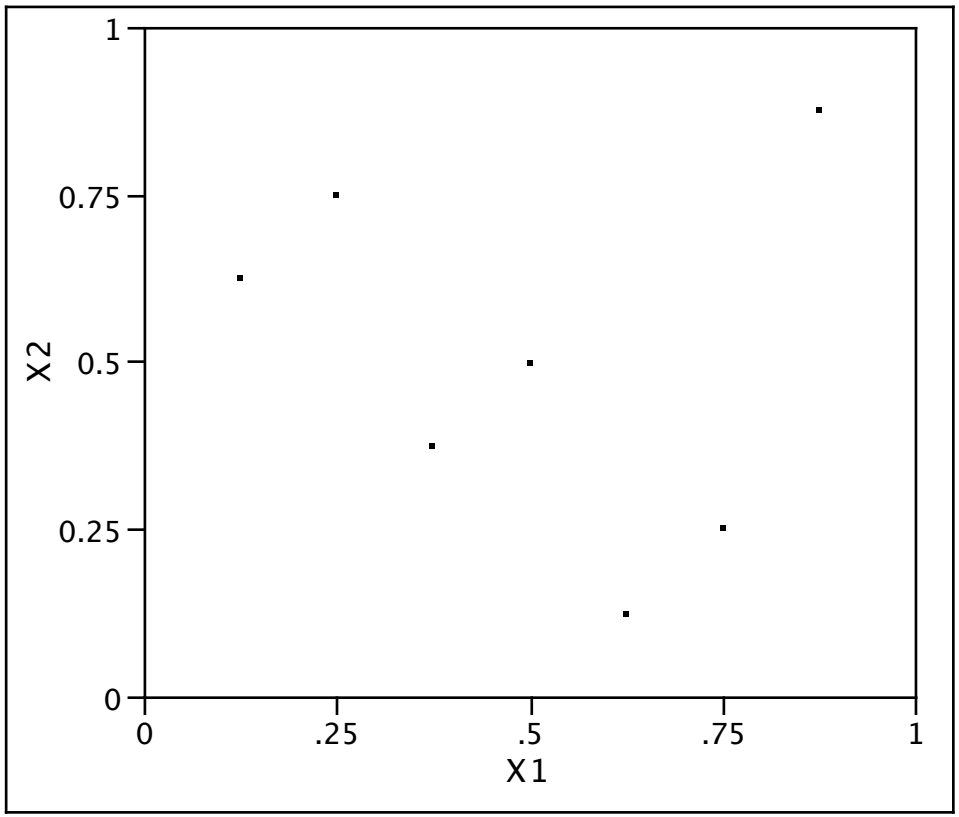
5 point Sobol in two dimensions



6 point Sobol in two dimensions



7 point Sobol in two dimensions



Some practical advice

1. Use a space-filling design. Our experience (based on several simulations) is that there is not much difference between space-filling designs. However, designs that are decidedly non space-filling do not perform well.

2. There are no analytical results on sample size. A popular rule of thumb is to take about 10 observations per dimension. Thus, a d dimensional problem would require $10d$ observations. We have found that fewer observations per dimension often suffices (say, 5 per dimension) unless one knows in advance that the output function is quite variable over the range of the inputs.

3. Sequential designs are useful if analysis can proceed sequentially and the computer code is slow. At each stage, while you wait for the code to generate the next output, you can carry out a preliminary analysis and decide if the results are sufficiently accurate.

Now, on to modeling