Emulators and Design

Andrew Parnell and Philip Cardiff
andrew.parnell@mu.ie

Maynooth University
National University of Ireland Maynooth

https://andrewcparnell.github.io/intro_emulators/

Introduction

- In this part of the course we will cover how to choose the input values at which to run our simulator
- Recall that the simulator is slow to run and we can only afford a small number of runs
- We need to choose the 'best possible' input values to run the simulator
- The values that we choose will become the inputs (features) to our machine learning model; the outputs will become the targets
- How do we choose these best values?

Choosing the total number of runs of the simulator

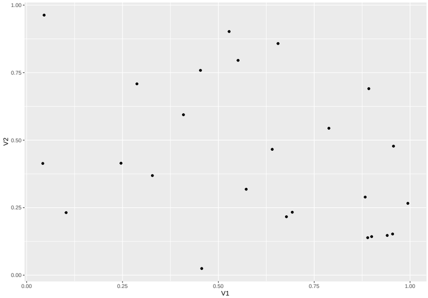
- Choosing the total number of runs is not really a mathematical/statistical problem, but it needs to be as large as possible
- It depends on the speed of the simulator, and how much patience/super computer time we have to spend on running it
- With some assumptions about the variability of the simulator surface it is possible to work out the expected uncertainty in the emulator for a given number of runs, but in the end it always comes down to doing as many as you possibly can

Choosing which input values to run the simulator at is a much harder problem. Here are some ideas. . .

Bad idea number 1 - random values

We could just choose random values across the input space:

```
N_sim <- 25
N_input <- 2
X <- matrix(runif(N_sim * N_input), ncol = N_input, nrow = N_sim)
```

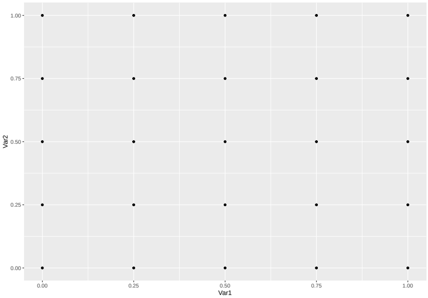


This is bad because we might miss large chunks of the input space; and it gets worse in higher dimensions

Bad idea number 2 - grids

Alternatively we could grid up the input space to cover the full region

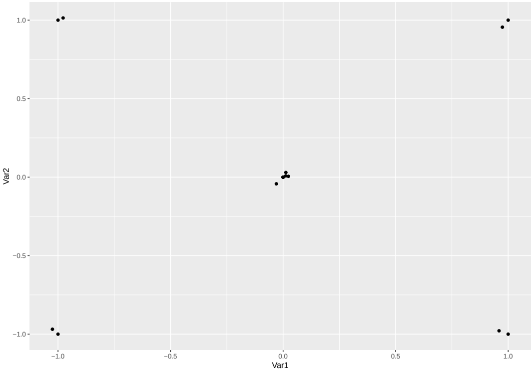
```
x <- seq(0, 1, length = sqrt(N_sim))
X <- expand.grid(x, x)
```



... but this is also bad because (a) one or more of the variables might not be important (so harder to identify non-linear effects)

Bad idea number 3 - traditional design

Here's a Central-Composite (similar to Box-Behnken) design

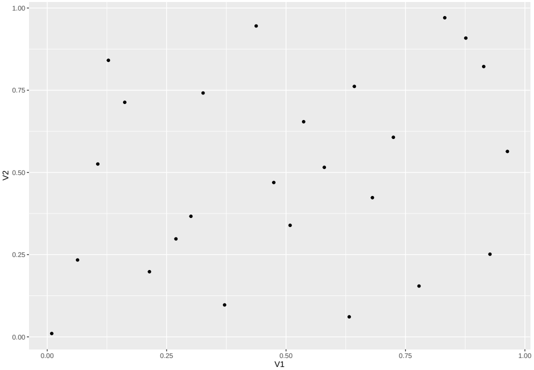


... but since the simulator is deterministic there is no point in running it at the same input points more than once

A better idea - Latin hypercubes

A better method is a Latin hypercube design

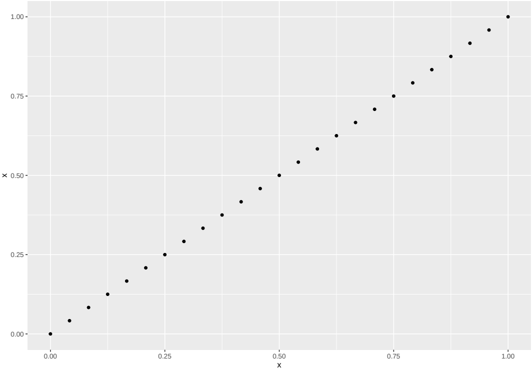
```
library(lhs)
X <- maximinLHS(N_sim, N_input)
```



Think of dividing up the input space into horizontal and vertical bands, and picking one one value that covers each row and each column

Bad Latin hypercubes

Actually, that idea doesn't work very well, because this is also a valid Latin hypercube design:

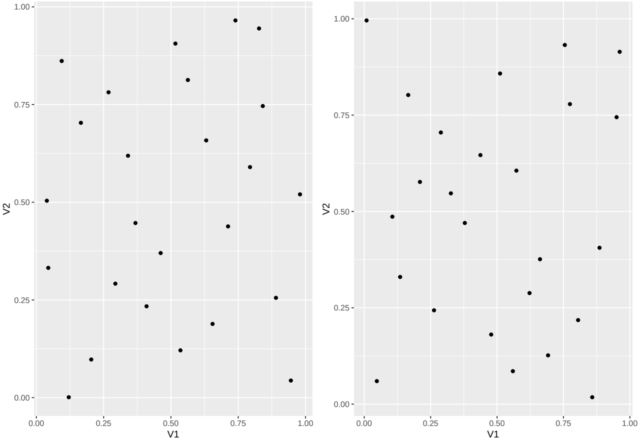


We want a design that is both a Latin hypercube and fills as much of the space as possible

Good Latin hypercubes

Try to opimise the sample by finding a Latin Hypercube sample that maximises the minimum distance between design points

We can generate lots of these with the function `maximinLHS`:



Designs for example 1

- Recall example 1: our simple sine wave with 2 inputs:

```
f <- function(x1, x2) {
  return(10 * sin(pi * x1 * x2))
}
```

Suppose we are willing to run this simulator 20 times. Create the design with:

```
n_runs <- 20
n_inputs <- 2
initial_grid <- maximinLHS(n_runs, n_inputs)
```

Designs for example 2

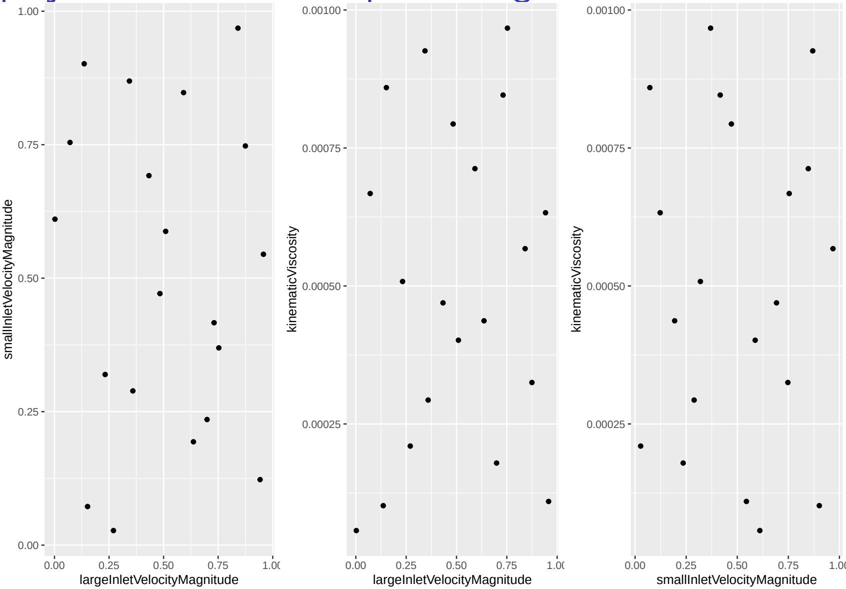
Recall example 2 is our 2D Navier Stokes model with 3 inputs

Suppose we are willing to run this 20 times:

```
n_runs <- 20
n_inputs <- 3
initial_grid <- maximinLHS(n_runs, n_inputs)
```

... harder to plot this as now in 3 dimensions

Some 2D projections of the example 2 design



Where to get more information on design

There are many possibilities when it comes to design in emulation:

- More advanced ways of spreading out the design and ensuring you do not miss important parts of the space
- More advanced ways of taking into account knowledge about the likely values of the inputs. For example, we might be able to guess a probability distribution for the input variables and use this to 'target' values in the design
- See Chapter 5 of The Design and Analysis of Computer Experiments for more detailed discussion
- We will stick to using Maximin Latin hypercube samples for our emulator

Summary

- We need to choose how many total runs we can afford
- Lots of traditional design ideas do not work well for emulator design
- Remember that the simulator is deterministic - not much point running at the same values twice! And try to avoid gridded values if you can
- Using Maximin Latin hypercube samples a good default choice
- Next: building the emulator...