Discussion: LHS

Today we'll use package lhs to do latin hypercube sampling.

As discussed in class, a latin hypercube is a method to sample parameter values. It is significantly more efficient than random sampling, so it was very important when computing was not as cheap and effective as it is today. Even with fast computing, it is still useful because LHS generates random numbers that are representative of the real variability for all parameters.

In simple random sampling, each point is sampled independently from all others, which means that your sample could be very clumped, or not have a single value that is in a big region of parameter space.

Technically, pure random sampling should still work, but with caveats: the convergence is inneficient, so you need to sample *a lot* of parameter sets and it is not obvious how many is *a lot*.

LHS forces your sampling to be more homogeneous by spreading out the sampled values to smaller intervals within the whole parameter range. A way to think of this is to imagine the parameter space partitioned in a square grid. The latin hypercube samples only one value per row/ and column of that grid.

> Exercise 1: install and load R package 'lhs'
>
> Exercise 2: look at the documentation for function randomLHS in package lhs. What does it do? What argu-

ments does it take?

Function randomLHS generates a hypercube sampled from a uniform distribution between 0 and 1. You can transform that output to fit your needs using the quantile distribution functions. For example, if one of your variables is normally distributed, you would use qnorm()

> Exercise 3: plot a randomLHS with 5 samples of 2 variables and 5x5 numbers sampled using runif. Show both to the person next to you. Can you tell which one is random and which one is latin? How about with 500 points?

Function randomLHS generates a hypercube sampled from a uniform distribution between 0 and 1. You can transform that output to fit your needs using the quantile distribution functions. For example, if one of your variables is normally distributed, you would use qnorm()

> Exercise 4: transform your output from exercise 3 using qnorm and plot the density of the points. Then plot the density of 50 numbers sampled from rnorm(). Can you tell which one is the latin cube now?
>
> Exercise 5: using the SIR model code from previous classes (you can copy it from my code at the end of this pdf too), generate a randomLHS matrix for the recovery rate and the transmissibility rate and run the model for each parameter combination, recording the time to peak outbreak. (Hint: you may want to use 2 nested for loops)
>
> Exercise 6: What is the influence of R_0 on time to peak outbreak? Plot these against each other. If we knew how long it took for the outbreak to peak would we be able to determine with certainty what R_0 is for this outbreak? (Ps: $R_0 = \frac{\beta}{\gamma}$ for the simple SIR model).

Exercise 7: Come up with a plot that shows this relationship and a plot that communicates how uncertainty on when the outbreak will peak given the range of values of β and γ you sampled. (Hint: a violin plot would be great).

```
####Example SIR code

require(deSolve)
S0 = 1-.0001   ####frequency dependent mode
I0 = .0001
R0 = 0
state_vars = c(SS = S0, II = I0, RR = R0)

#Generate a  series of times for the ODE solver
tseq <- seq(0, 60, by = .1)
#Generate a vector of parameter values
pars <- c(beta = 1.4247, gamma = 0.14)

SIR_system <- function(tseq, state_vars, pars){
    with(as.list(c(state_vars,pars)), {
        dS_dt = + mu*(SS+II+RR) - beta*SS*II - mu*SS
        dI_dt = beta*SS*II - gamma*II - mu*II
        dR_dt = gamma*II - mu*RR
        ###it is not necessary to have R in
        return(list(c(dS = dS_dt, dI = dI_dt, dR = dR_dt)))
    })
}

outputOriginal <- lsoda(state_vars, tseq, SIR_system, pars)
```