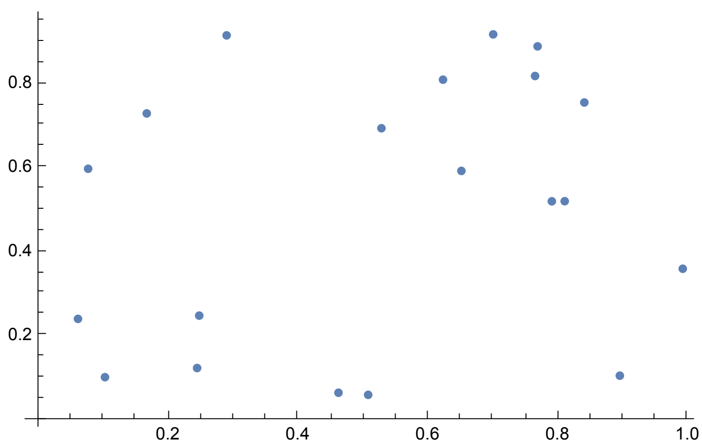
# Sets for Searching

```
num = 2001;
numv = {20, 50, 100, 200, 500, 1000, 2000};
```

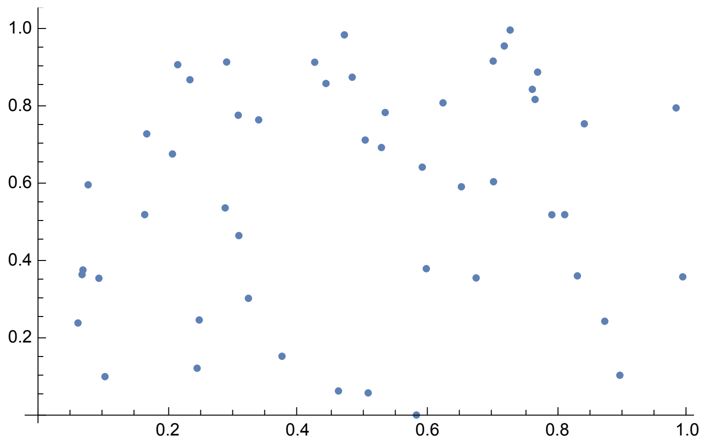
## Monte Carlo Sequences

```
dim1 = Table[Random[], {i, 1, num}];
dim2 = Table[Random[], {i, 1, num}];
Do[num = numv[[i]];
 mc = Table[{dim1⟦i⟧, dim2⟦i⟧}, {i, 1, num}];
Print[numv[[i]], " Monte Carlo points"];
 ListPlot[mc, AxesOrigin → {0, 0}] // Print,
 {i, 1, Length[numv]}]
```

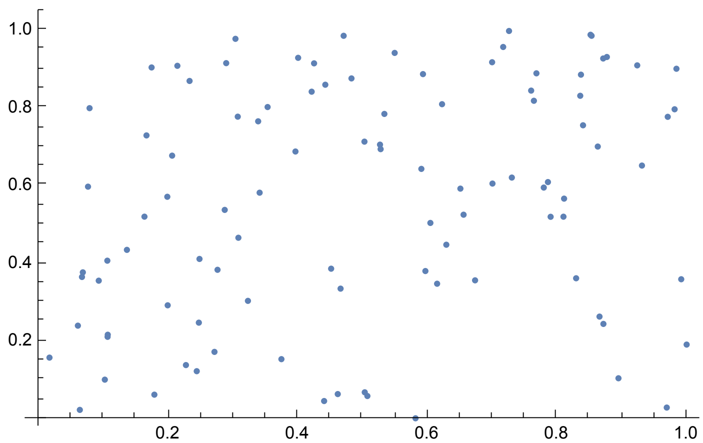
20 Monte Carlo points



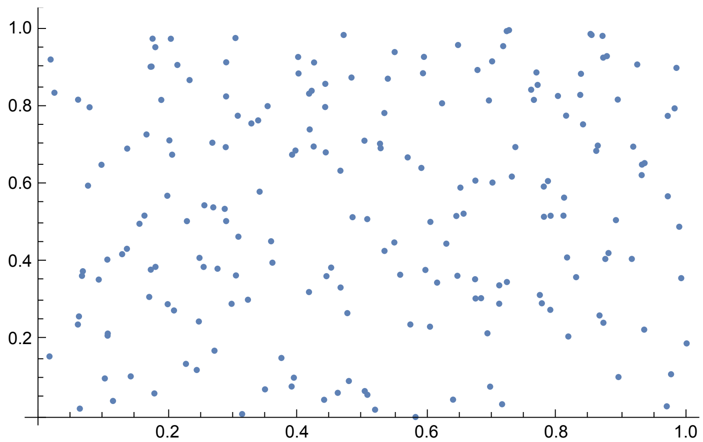
50 Monte Carlo points



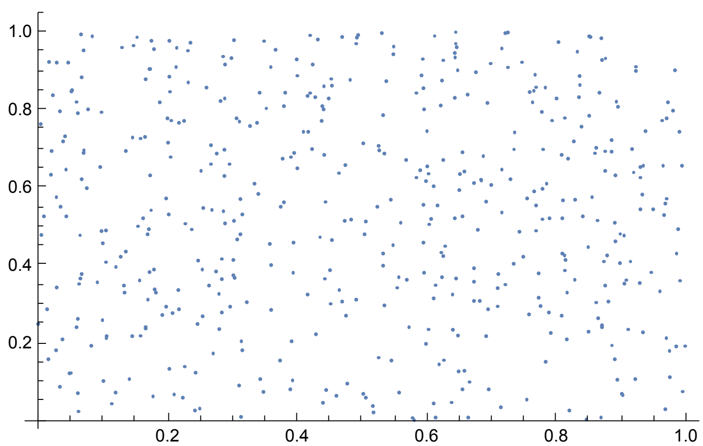

100 Monte Carlo points



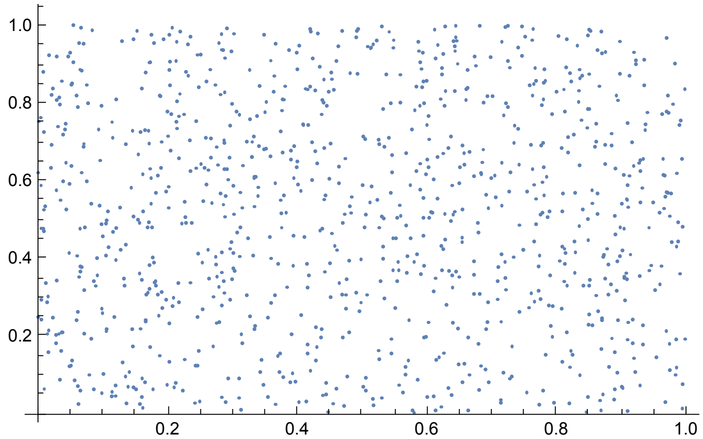
200 Monte Carlo points



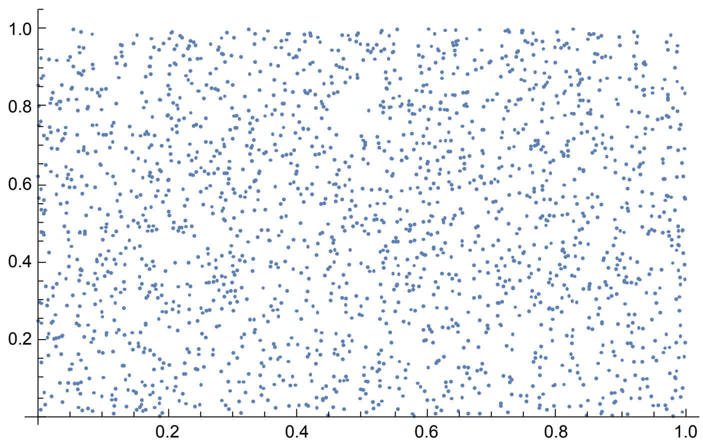

500 Monte Carlo points



```
1000 Monte Carlo points
```



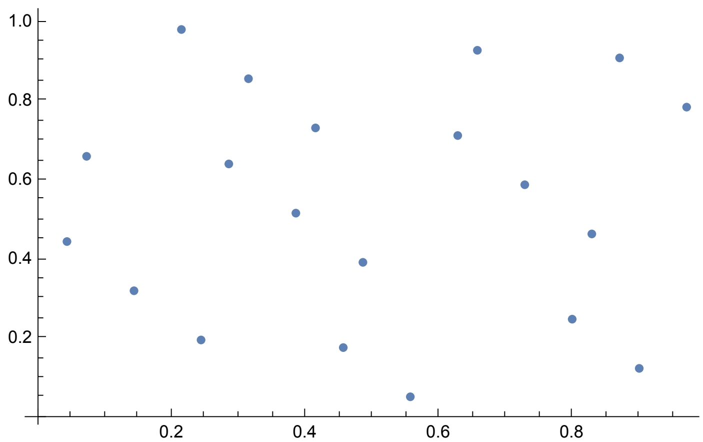

```
2000 Monte Carlo points
```



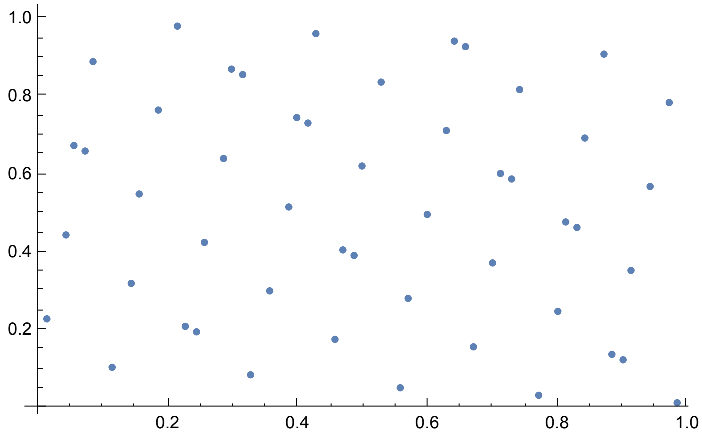
# Weyl Sequences

```
Do[num = numv[[i]];
 list = Table[i, {i, 1, num}];
dim1 = list 2^.5;
dim1 = dim1 - Floor[dim1];
dim2 = list 3^.5;
dim2 = dim2 - Floor[dim2];
weyl = Table[{dim1[[i]], dim2[[i]]}, {i, 1, num}];
Print[numv[[i]], " Weyl points"];
 ListPlot[weyl, AxesOrigin → {0, 0}] // Print,
 {i, 1, Length[numv]}]
```

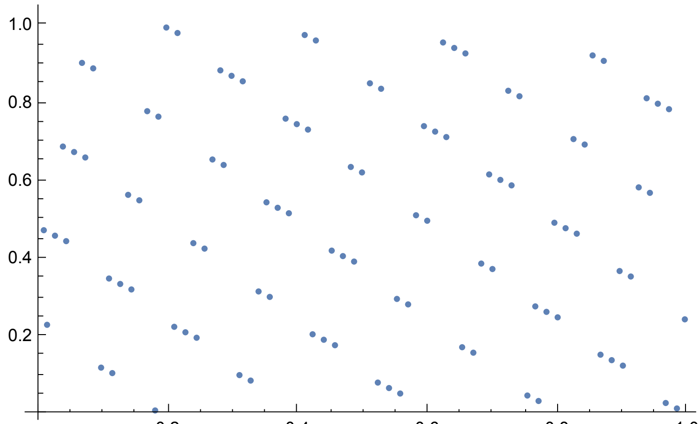
```
20 Weyl points
```



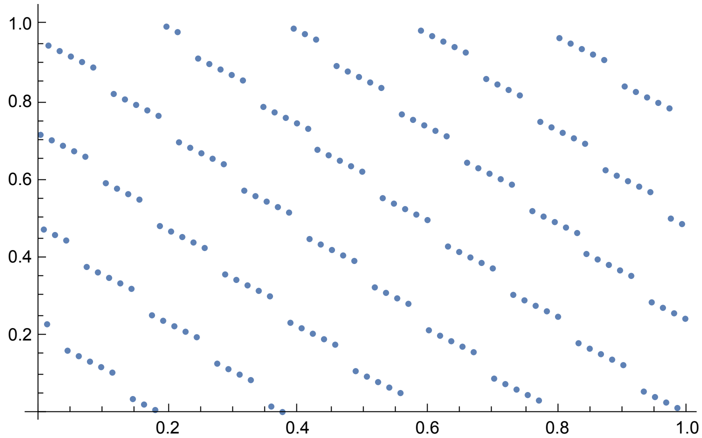
50 Weyl points



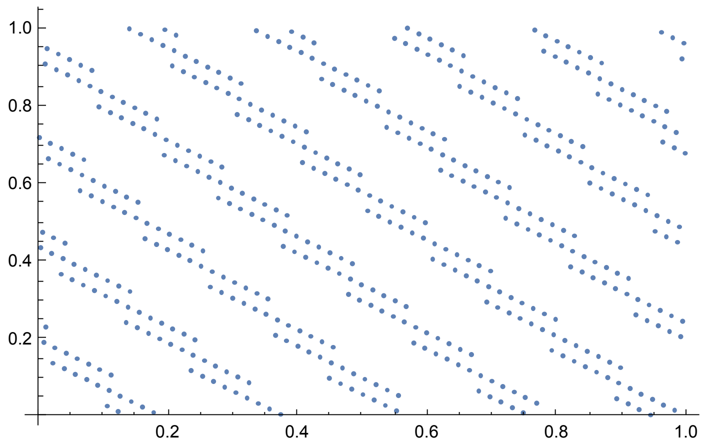

100 Weyl points



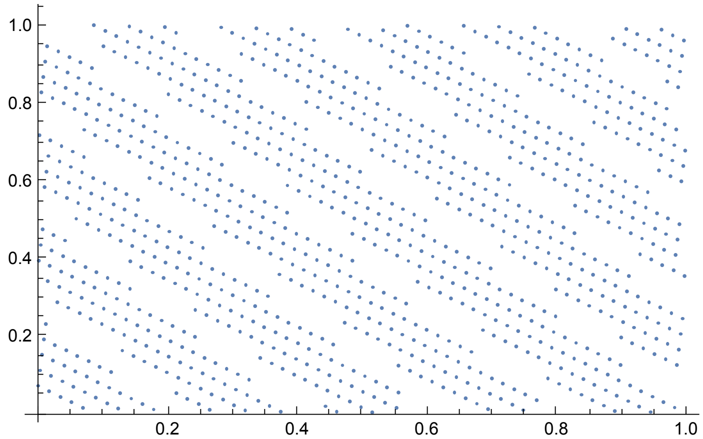
200 Weyl points



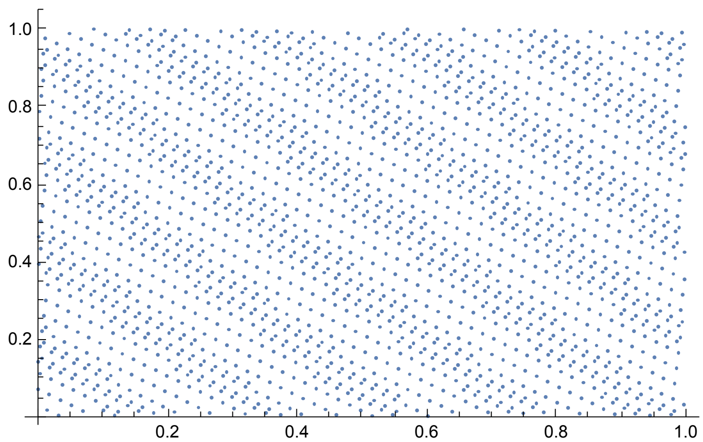

500 Weyl points



1000 Weyl points



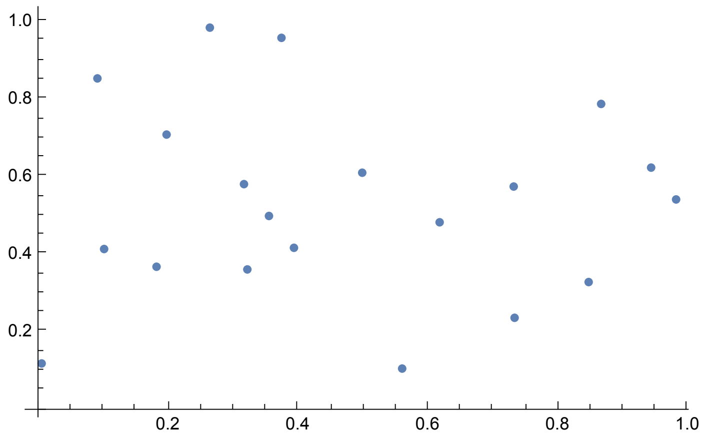

2000 Weyl points



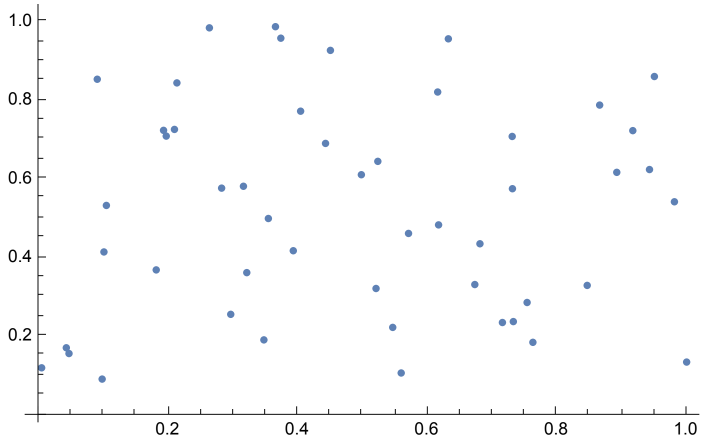
# Haber Sequences

```
Do[
 num = numv[[i]];
 list = Table[i (i + 1) / 2, {i, 1, num}];
dim1 = N[list 3^.5];
dim1 = dim1 - Floor[dim1];
dim2 = N[list 5^.5];
dim2 = dim2 - Floor[dim2];
 nnn = Table[{dim1⟦i⟧, dim2⟦i⟧}, {i, 1, num}];
Print[numv[[i]], " Haber points"];
 ListPlot[nnn, AxesOrigin → {0, 0}] // Print,
 {i, 1, Length[numv]}]
```

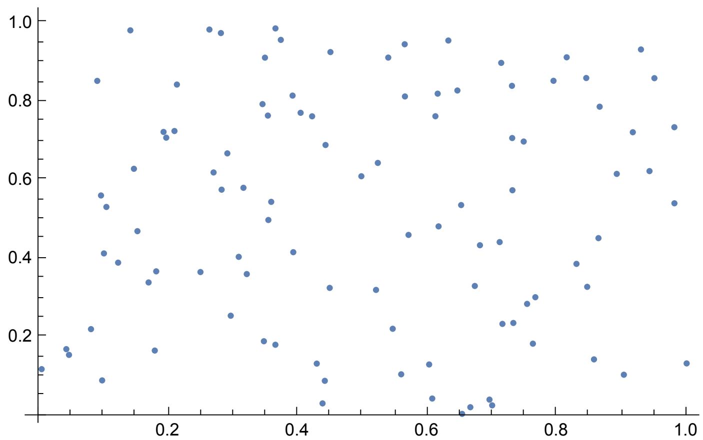
```
20 Haber points
```



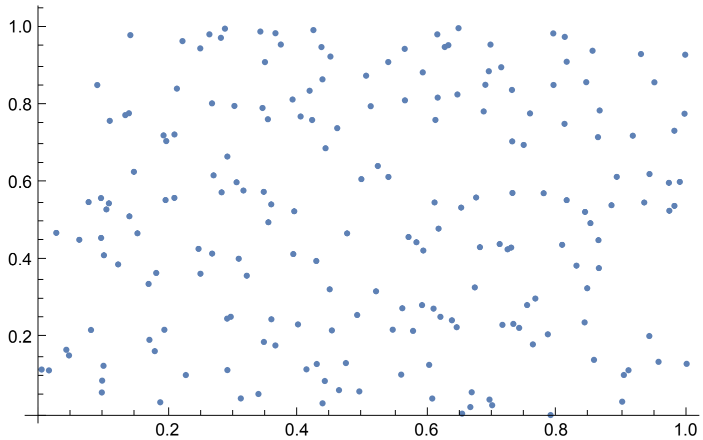
50 Haber points



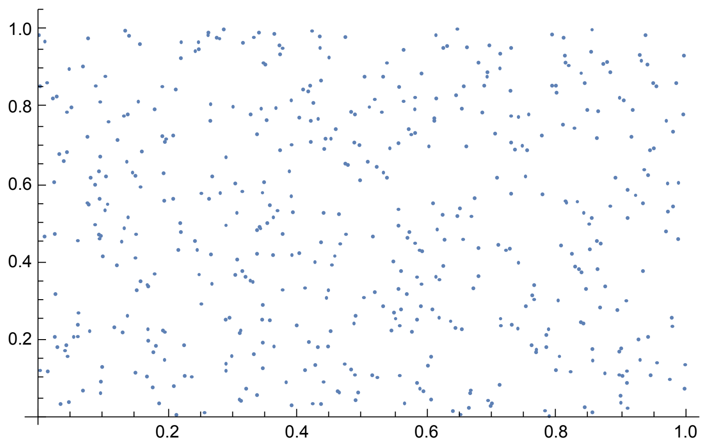

100 Haber points



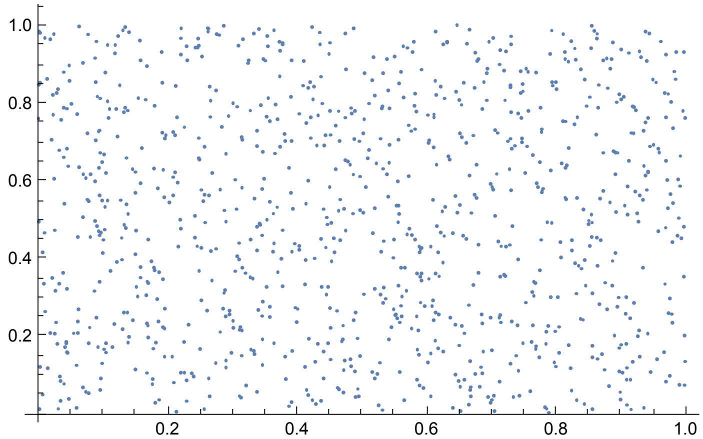
```
200 Haber points
```



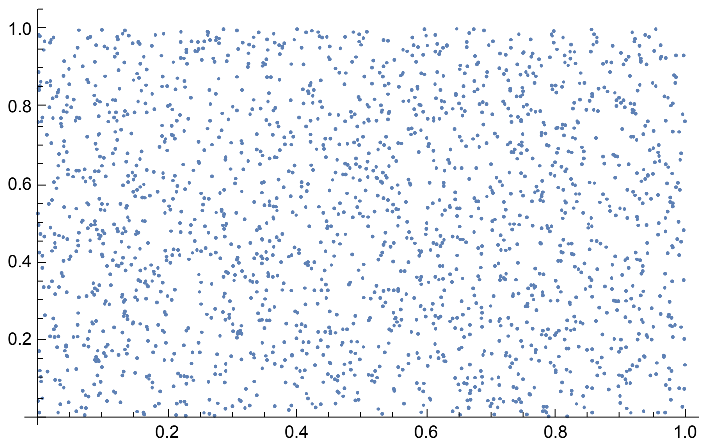

```
500 Haber points
```



1000 Haber points



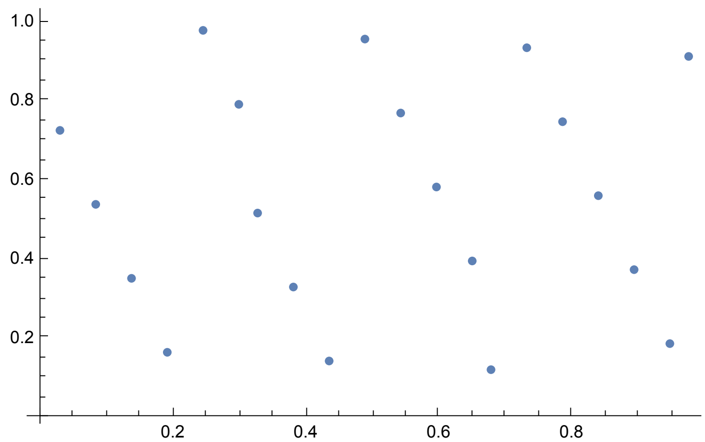

2000 Haber points



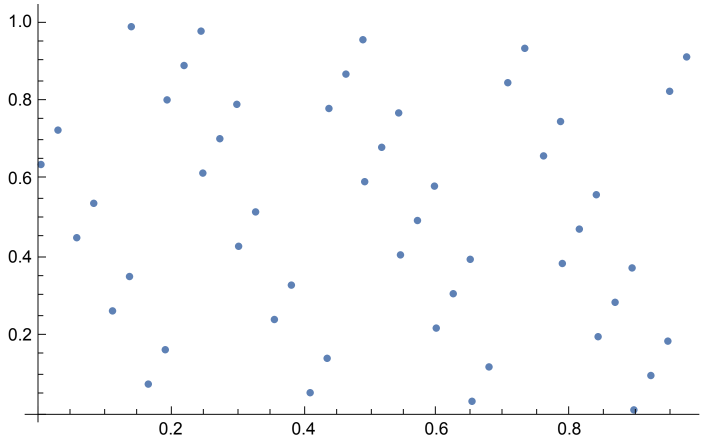
# Baker Sequences

```
Do[
 num = numv[[i]];
 list = Table[i, {i, 1, num}];
r1 = 1 / 2; r2 = 1 / 3;
dim1 = N[list Exp[r1]];
dim1 = dim1 - Floor[dim1];
dim2 = N[list Exp[r2]];
dim2 = dim2 - Floor[dim2];
 nnn = Table[{dim1⟦i⟧, dim2⟦i⟧}, {i, 1, num}];
Print[numv[[i]], " Baker points"];
 ListPlot[nnn, AxesOrigin → {0, 0}] // Print,
 {i, 1, Length[numv]}]
```

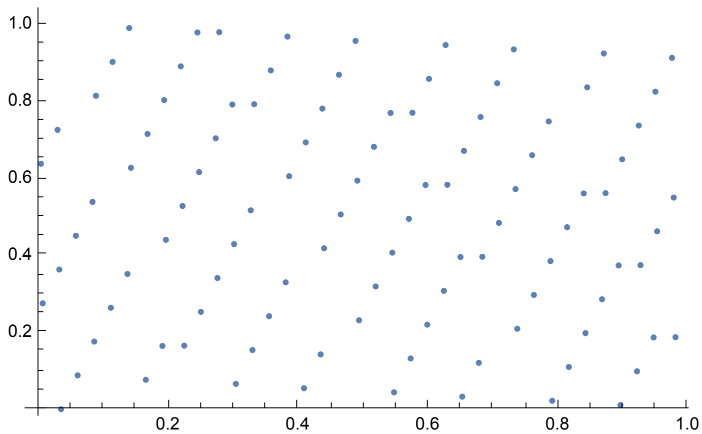
20 Baker points



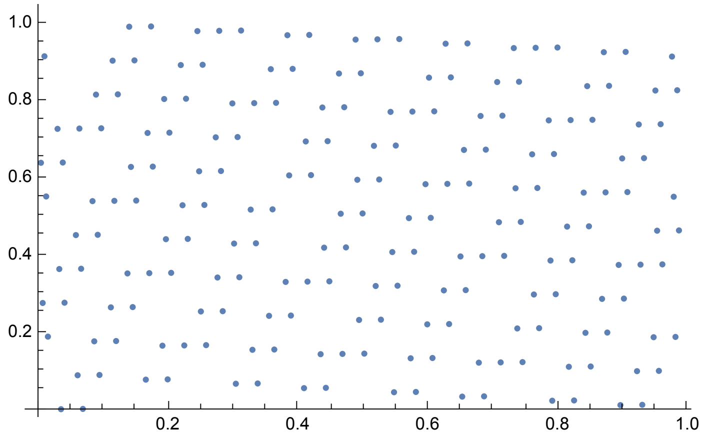
50 Baker points



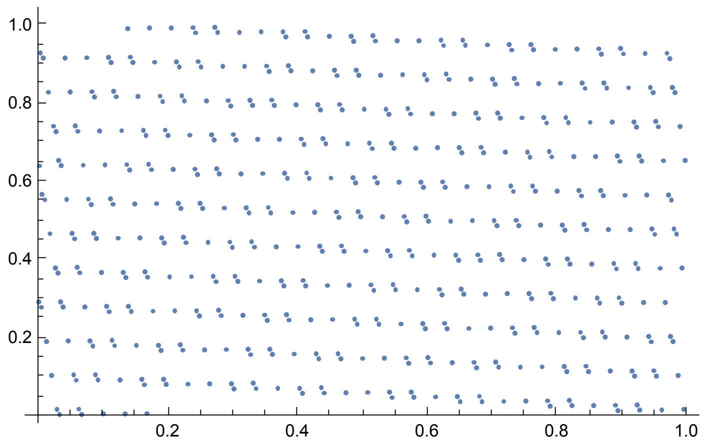

100 Baker points



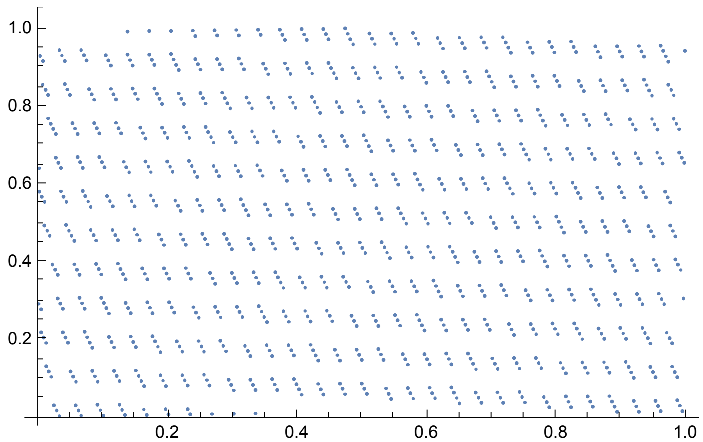
```
200 Baker points
```



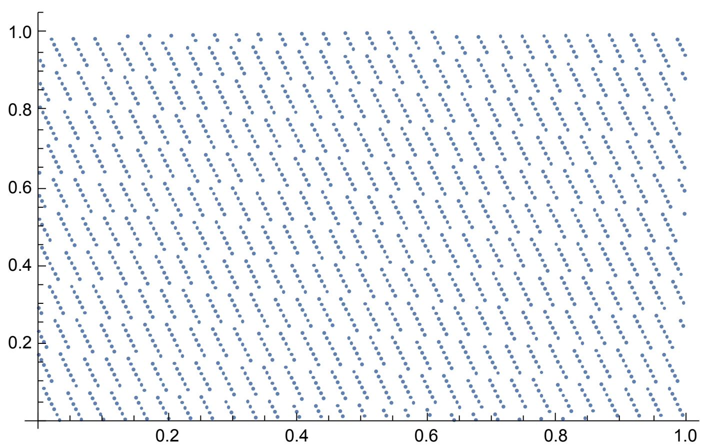

```
500 Baker points
```



1000 Baker points




2000 Baker points

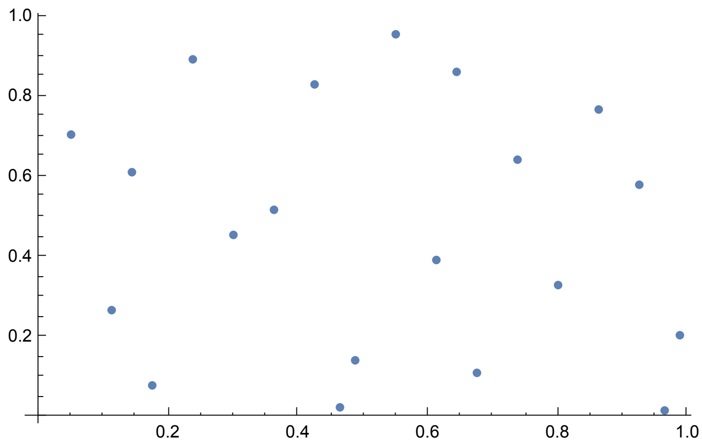


# Sobol Sequences

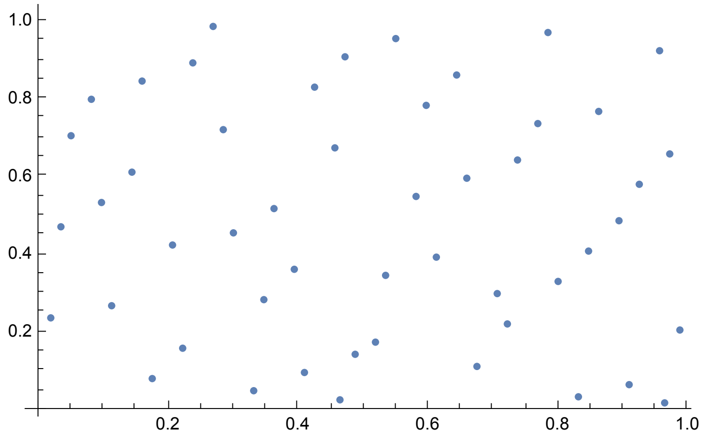
```
SeedRandom[Method -> {"MKL", Method -> {"Niederreiter", "Dimension" -> 2}}];
pts = RandomReal[1, {2000, 2}];
```

```
Do[Print[k, " Sobol points"];
ListPlot[pts[[1 ;; k]]] // Print, {k, numv}]
```

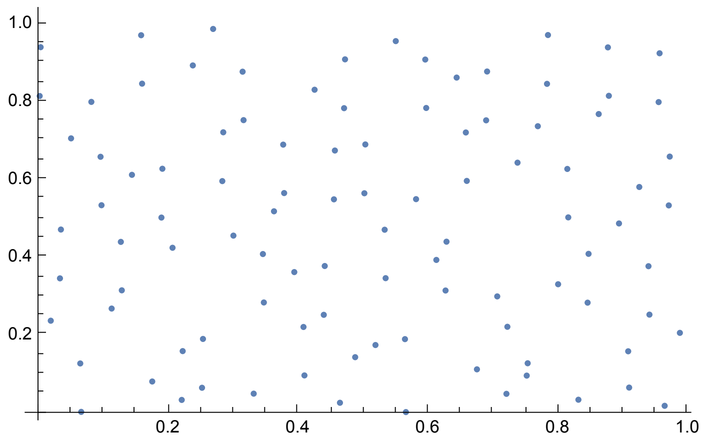
20 Sobol points



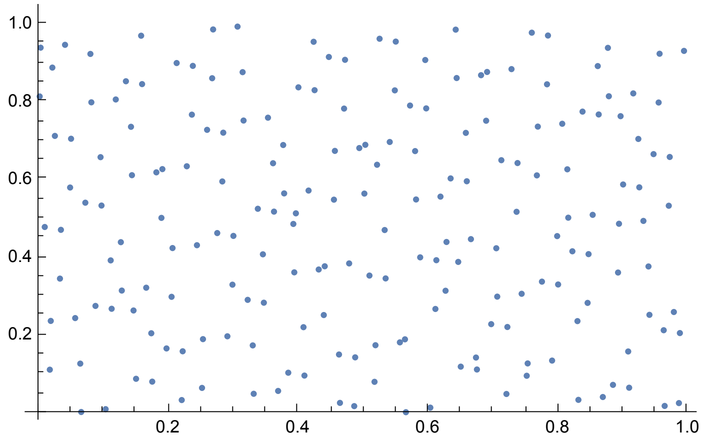
```
50 Sobol points
```



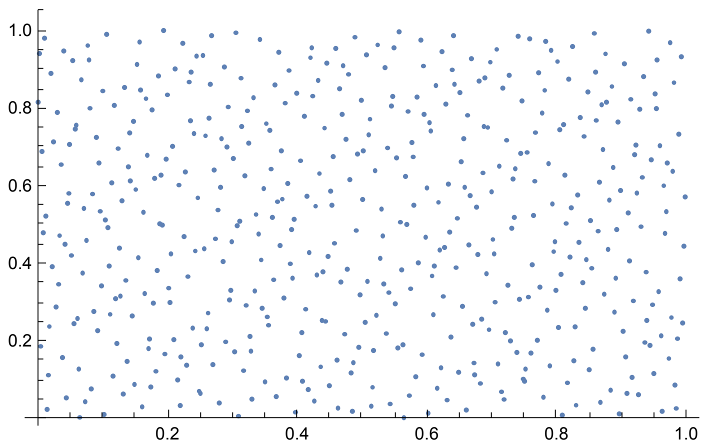

```
100 Sobol points
```



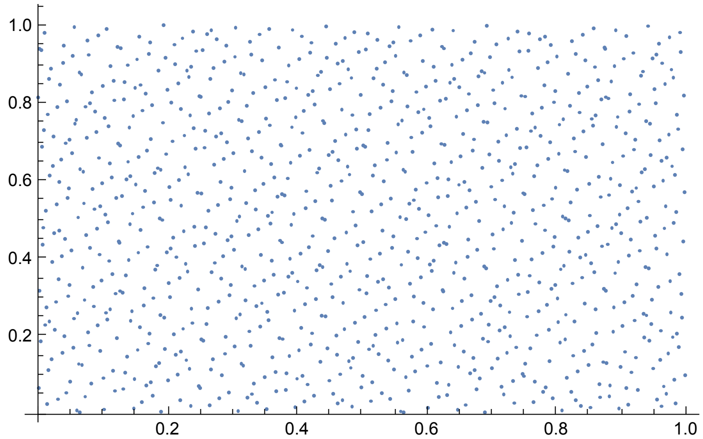
200 Sobol points



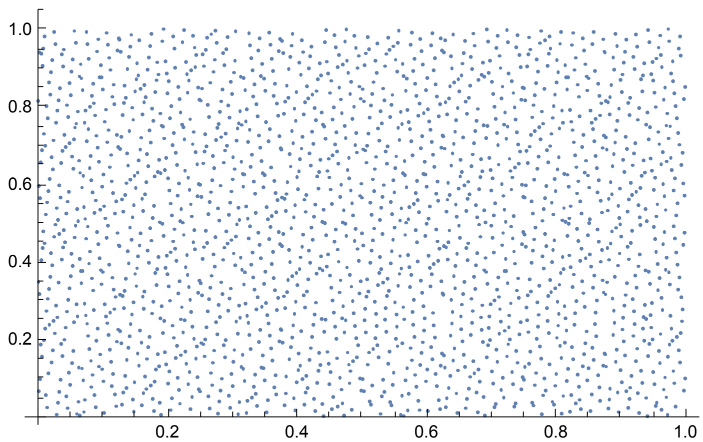

500 Sobol points



1000 Sobol points




2000 Sobol points



# Monomial Quadrature Rules

- Designed to be good quadrature rule
- Will also be good for finding good polynomial approximation
- Large collection of known rules
- Easy to construct new ones IF one has access to solvers for polynomial equations
    - Bertini -- homotopy continuation method
    - Groebner basis -- use modular method to exploit parallelism
- IDEA: Use computer power to find good sets
    - Use massive parallelism
    - Fixed cost; need not compute it again

# Korobov-Keast sets

- Designed to create sets of fixed size with low discrepancy
    - No asymptotics
    - Can compute the discrepancy
    - Can find computationally
- IDEA: Use computer power to find good sets
    - Use massive parallelism
    - Fixed cost; need not compute it again