# Are fuzzy dice intransitive?

With many sorts of discrete-valued dice, ties are possible. However, if we use "dice" which return a real number in [0,1] with each roll, then ties have measure zero.

There presumably are many ways to make such a function. Here, we use a Gaussian Process ("GP"). This is sort of a probability distribution over functions from R to R[1].

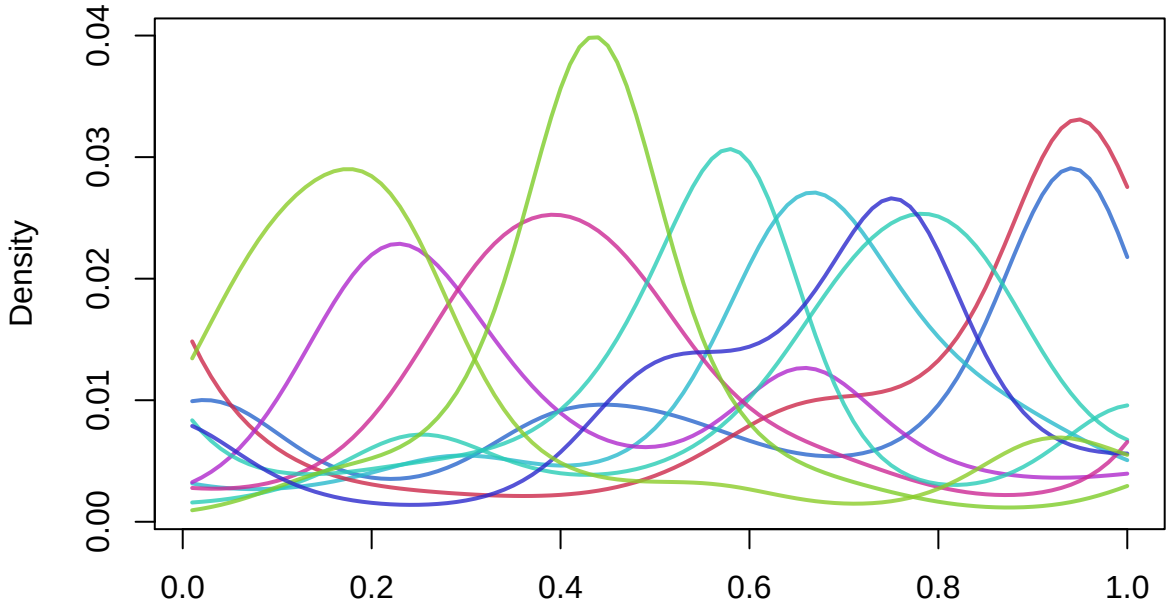
Of course, in a program, we can only measure the function at a finite subset of $x$ values. However, we can do this for as many points as we wish. (If we want to "zoom in" to sample at higher resolution, we could condition on the points we've already sampled). In order to make this a positive function, we exponentiate it. To make it a proper probability density function, we then normalize it.

(Note: this code is written in R, and is only occasionally written in an efficient way).

```
library(mvtnorm)   # FIXME use stat package version of this?
gp.dice = function(n = 100, num.dice = 1000, length.scale = 0.2) {
  x = c(1:n) / n
  d = (outer(x, x, "-") / length.scale)^2
  s = exp(-d)
  y1 = rmvnorm(num.dice, mean=rep(0,n), sigma=s)
  y = exp(y1)
  y = y / apply(y, 1, sum)
  y
}
```

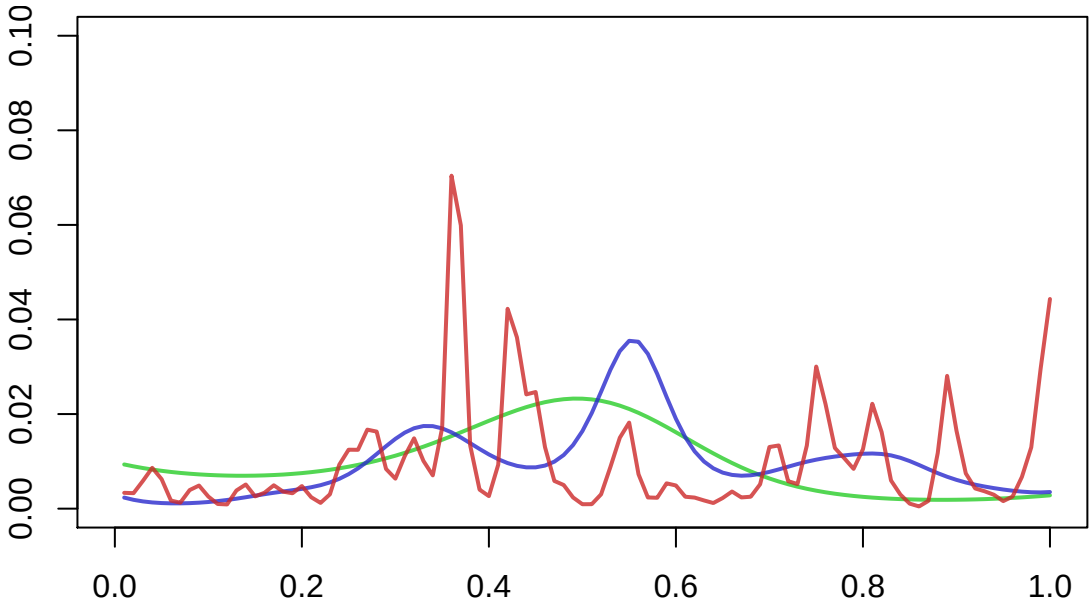
```
set.seed(123)
n = 100
x = c(1:n) / n
y = gp.dice(n=n)
ylim = range(y[1:10,])
plot(x,y[1,], ylim=ylim, type="n", xlab="", ylab="Density")
for(i in 1:10)
  lines(x, y[i,], col=hsv(runif(1), 0.8, 0.8, 0.8), lwd=2,
    xlab="dice value", ylab="probability")
```

[1] IIRC, Hanna Wallach's GP tutorial described this as "sampling directly from the function space", but I can't find those notes online anymore.



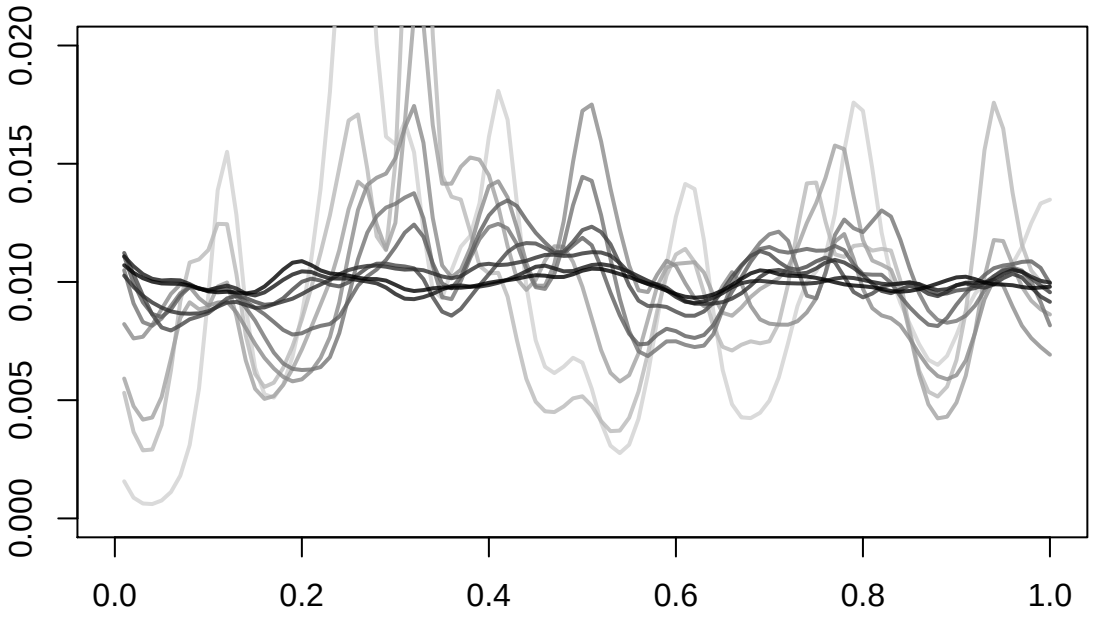

Changing the "length scale" of this changes how smooth the distributions are.

```
ylim=c(0, 0.1)
length.scale = c(0.3,0.1,0.02)
plot(1,1, xlim=c(0,1), ylim=ylim, type="n", xlab="", ylab="")
for(i in 1:3) {
  y1 = gp.dice(n=100, num.dice=3, length.scale=length.scale[i])
  range(y1)
  for(j in 1:1) {
    lines(x, y1[j,], ylim=c(0, max(y1)), col=hsv(i/3, 0.8, 0.8, 0.8),lwd=2)
  }
}
```




The overall distribution of these dice may or may not be uniform. Here, we show the average of (increasing numbers of) these dice:

```
y = gp.dice(n=100, num.dice=1024, length.scale=0.05)
plot(1,1, xlim=c(0,1), ylim=c(0, 2*mean(y)), type="n", xlab="", ylab="")
for(i in 1:10) {
  y1 = apply(y[1:(2^i),], 2, mean)
  lines(x, y1, col=hsv(0, 0, (10-i)/11, 0.8),lwd=2)
}
```




```
# only use first few dice for further simulations
y = y[1:200,]
```

However, P. Peng noted that any monotonically increasing transformation of the dice values won't change which dice win. Presumably, this means that we could assume that the dice converge to a chosen distribution (uniform, or normal, say) by picking an appropriate transformation (although I don't know what this GP converges to, on average).

## Running a tournament

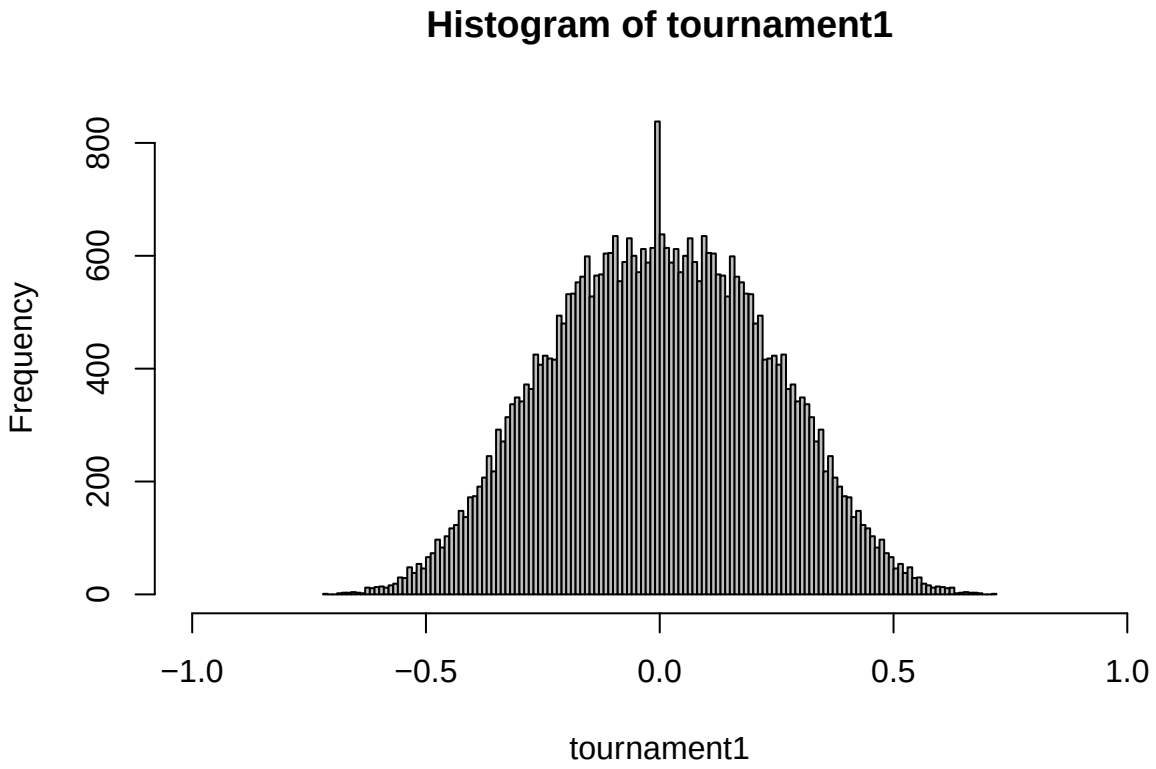
We can then run a tournament amongst these dice. We compare the probability of A being higher than B, ignoring ties. (We could also compute the expected value of this difference).

```
# Given a matrix of dice, computes the probability that each die beats
# each other die.
#   dice.dist: the distribution of the dice, with one row per die
#   Returns: matrix of the probability of A beating B
dice.tournament = function(dice.dist) {
  n = ncol(dice.dist)
  # compute a matrix with 1 in the upper triangle, -1 in the lower
  # triangle, and 0 on the diagonal
  s = matrix(NA, n, n)
  for(i in 1:n)
    for(j in 1:n)
      s[i,j] = sign(i - j)
```

```
  num.dice = nrow(dice.dist)
  r = matrix(NA, num.dice, num.dice)
  for(i in 1:num.dice)
    for(j in 1:num.dice) {
      # compute probability of each outcome, and add them up
      p = outer(dice.dist[i,], dice.dist[j,])
      r[i,j] = sum(p * s)
    }
  r
}
tournament1 = dice.tournament(y)
```

We can look at the overall distribution of these tournament results:

```
hist(tournament1, xlim=c(-1,1), breaks=200, col="grey")
```




We can now look to see how often, if A beats B and B beats C, A beats C.

```
test.transitivity = function(tournament.result) {
  # binarize this, for simplicity
  A = tournament.result > 0
  n = nrow(tournament.result)
  n.AB.and.BC = 0
  n.AC = 0
  n.CA = 0
  # no doubt this could be done faster
  for(i in 1:n)
    for(j in 1:n)
```

```
      for(k in 1:n)
        if ((i!=j) && (j!=k) && (i!=k) && (A[i,j]) && (A[j,k])) {
          n.AB.and.BC = n.AB.and.BC + 1
          if (A[i,k])
            n.AC = n.AC + 1
          if (A[k,i])
            n.CA = n.CA + 1
        }

  c(n.AB.and.BC = n.AB.and.BC, n.AC = n.AC, n.CA = n.CA, mean.A = mean(A))
}
test.transitivity(tournament1)
```

```
## n.AB.and.BC        n.AC        n.CA      mean.A
##   1323392.0   1308404.0     14988.0         0.5
```

It appears that dice, defined this way, are very transitive.

## Centering the dice

The way the dice are currently defined doesn't force them to be "centered" in any way. One of the dice definitions currently popular in the Polymath thread is (I believe) dice which add up to a particular number.

An analogous idea, with these dice, might be to center their expected value at 0. Perhaps such dice are more likely to be intransitive. The way I'm currently doing this is somewhat hacky.

```
# Computes expected value of a function, expected to be a
# probability density on equally-spaced points in [0,1].
expected.value = function(p) {
  p = p / sum(p)  # normalize; shouldn't be needed
  n = length(p)
  sum( p * c(1:n) / n )
}
# Deprecated.
reweight.dist.deprecated.1 = function(p) {
  n = length(p)
  i = trunc(n * expected.value(p) + 0.5)
  if (i<0) i = 0
  if (i>n) i = n
  # pad the original distribution with zeros
  z = function(num.zeros) if (num.zeros==0) c() else rep(0, num.zeros)
  r = c(z(n-i), p, z(i))
  r / sum(r)
}
# Deprecated.
reweight.dist.deprecated.2 = function(p) {
  n = length(p)
  a = 1:n
  b = n:1
  a.s = sum(a * p)
  b.s = sum(b * p)
  r = p * (a.s * b + b.s * a)
```

```
  r / sum(r)
}
# Reweights a distribution to have expected value 0.5 .
reweight.dist = function(p) {
  n = length(p)
  i = c(1:n) / n
  f = function(m) {
    (expected.value(p*exp(m*i)) - 0.5)^2
  }
  m = optimize(f, c(-30, 30))$minimum
  r = p*exp(m*i)
  r / sum(r)
}
y.centered = t(apply(y, 1, reweight.dist))
```

We now re-run the tournament with the centered dice.

```
tournament2 = dice.tournament(y.centered)
test.transitivity(tournament2)
```

```
##  n.AB.and.BC         n.AC         n.CA       mean.A
## 1.963434e+06 9.883830e+05 9.750510e+05 5.002250e-01
```

The centered dice appear more transitive.

## Summary

It is possible to sample a function defined on, at least, a countably-infinite set of points. (I'm not sure how to sample a continuous function from [0,1]->R).

According to a simulation (not sampling an actual probability density defined on [0,1] -> R, but just on a grid of points), such dice aren't necessarily transitive.

However, if we (somewhat hackily) force such dice to have the same mean, they (superficially appear) transitive.

I don't know what consequences this has for the larger Polymath intransitive-dice question.