Finding the collision probabilities

K is the number of balls (random numbers). M is the number of urns. (K<M)
I use the following parameters.

In[1]:=
```
M = 1000;
K = 200;
```

Below I will write a code to implement the recursion discussed in class.

First I define the initial conditions for the probabilities p[j, n] discussed in the lecture.

In[3]:=
```
p[1, 1] = 1.;
```

In[4]:=
```
Do[p[1, n] = M^(1. - n), {n, 2, K}];
```

These are the "diagonal" values:

In[5]:=
```
Do[p[n, n] = (M - n + 1.)/M * p[n - 1, n - 1], {n, 2, K}]
```

Now I code the basic recursion:

In[6]:=
```
Do[
 Do[p[j, n] = (j/M) * p[j, n - 1] + (M - j + 1.)/M * p[j - 1, n - 1], {n, j + 1, K}],
 {j, 2, K}]
```

Most of these probabilities are actually very small. There are only relatively small number of "nonzero" values. This feature is essential and it needs to be exploited if we want to write an efficient code that can deal with large data. See Knuth for a discussion of this.

Now I compute the probabilities of collisions, p[i,K], i = 1,...,K. Recall that p[i,K] gives the probability that there are K-i collisions when M balls are tossed. I store these values in the list pr.

In[7]:=
```
pr = Table[p[i, K], {i, 1, K}]
```

Out[7]=
$$\{1.000000000000\times10^{-597}, 8.026655531073676\times10^{-535}, 4.413627968953761\times10^{-497},$$
$$1.069493653363979\times10^{-469}, 5.134168849679878\times10^{-448}, 5.839708745556878\times10^{-430},$$
$$2.032532308812555\times10^{-414}, 1.000659020800411\times10^{-400}, 1.875252559083344\times10^{-388},$$
$$2.634095586035953\times10^{-377}, 4.502057625391830\times10^{-367}, 1.340102921157190\times10^{-357},$$
$$9.12793444354448\times10^{-349}, 1.759935332847609\times10^{-340}, 1.137424658554164\times10^{-332},$$
$$2.824704407584983\times10^{-325}, 3.014977793744853\times10^{-318}, 1.517878881516296\times10^{-311},$$
$$3.89702\times10^{-305}, 5.45145\times10^{-299}, 4.397\times10^{-293}, 2.14709\times10^{-287}, 6.62196\times10^{-282}, 1.33854\times10^{-276},$$
$$1.83188\times10^{-271}, 1.74685\times10^{-266}, 1.19062\times10^{-261}, 5.93361\times10^{-257}, 2.2066\times10^{-252}, 6.2363\times10^{-248},$$
$$1.36172\times10^{-243}, 2.33178\times10^{-239}, 3.174\times10^{-235}, 3.47696\times10^{-231}, 3.09996\times10^{-227}, 2.27275\times10^{-223},$$
$$1.38323\times10^{-219}, 7.04951\times10^{-216}, 3.03271\times10^{-212}, 1.1095\times10^{-208}, 3.47557\times10^{-205}, 9.3819\times10^{-202},$$
$$2.19527\times10^{-198}, 4.47724\times10^{-195}, 7.99987\times10^{-192}, 1.25832\times10^{-188}, 1.75016\times10^{-185},$$
$$2.16157\times10^{-182}, 2.38\times10^{-179}, 2.34482\times10^{-176}, 2.07433\times10^{-173}, 1.65313\times10^{-170}, 1.19053\times10^{-167},$$
$$7.77039\times10^{-165}, 4.60911\times10^{-162}, 2.49112\times10^{-159}, 1.22984\times10^{-156}, 5.55905\times10^{-154},$$
$$2.30576\times10^{-151}, 8.79438\times10^{-149}, 3.09064\times10^{-146}, 1.0027\times10^{-143}, 3.00862\times10^{-141}, 8.36348\times10^{-139},$$
$$2.15749\times10^{-136}, 5.17292\times10^{-134}, 1.15453\times10^{-131}, 2.40204\times10^{-129}, 4.6651\times10^{-127}, 8.46868\times10^{-125},$$
$$1.43878\times10^{-122}, 2.29045\times10^{-120}, 3.42056\times10^{-118}, 4.79738\times10^{-116}, 6.32565\times10^{-114},$$
$$7.84953\times10^{-112}, 9.1758\times10^{-110}, 1.01138\times10^{-107}, 1.05209\times10^{-105}, 1.03378\times10^{-103}, 9.60309\times10^{-102},$$
$$8.44007\times10^{-100}, 7.02377\times10^{-98}, 5.53869\times10^{-96}, 4.14158\times10^{-94}, 2.93865\times10^{-92}, 1.97988\times10^{-90},$$
$$1.26742\times10^{-88}, 7.71356\times10^{-87}, 4.46584\times10^{-85}, 2.461\times10^{-83}, 1.29157\times10^{-81}, 6.45884\times10^{-80},$$
$$3.07923\times10^{-78}, 1.40021\times10^{-76}, 6.07597\times10^{-75}, 2.51713\times10^{-73}, 9.95988\times10^{-72}, 3.76569\times10^{-70},$$
$$1.36099\times10^{-68}, 4.70383\times10^{-67}, 1.55525\times10^{-65}, 4.92108\times10^{-64}, 1.49066\times10^{-62}, 4.32419\times10^{-61},$$
$$1.20164\times10^{-59}, 3.19979\times10^{-58}, 8.16724\times10^{-57}, 1.99875\times10^{-55}, 4.69123\times10^{-54}, 1.05627\times10^{-52},$$
$$2.28206\times10^{-51}, 4.73201\times10^{-50}, 9.41956\times10^{-49}, 1.80041\times10^{-47}, 3.3049\times10^{-46}, 5.82735\times10^{-45},$$
$$9.87166\times10^{-44}, 1.6069\times10^{-42}, 2.51382\times10^{-41}, 3.78002\times10^{-40}, 5.46419\times10^{-39}, 7.59428\times10^{-38},$$
$$1.01491\times10^{-36}, 1.30435\times10^{-35}, 1.61224\times10^{-34}, 1.91678\times10^{-33}, 2.19207\times10^{-32}, 2.41161\times10^{-31},$$
$$2.55242\times10^{-30}, 2.59903\times10^{-29}, 2.54624\times10^{-28}, 2.40008\times10^{-27}, 2.1767\times10^{-26}, 1.89941\times10^{-25},$$
$$1.59472\times10^{-24}, 1.2882\times10^{-23}, 1.00116\times10^{-22}, 7.4856\times10^{-22}, 5.38431\times10^{-21}, 3.72549\times10^{-20},$$
$$2.47944\times10^{-19}, 1.58708\times10^{-18}, 9.7695\times10^{-18}, 5.78261\times10^{-17}, 3.29074\times10^{-16}, 1.80019\times10^{-15},$$
$$9.46507\times10^{-15}, 4.78226\times10^{-14}, 2.32146\times10^{-13}, 1.08246\times10^{-12}, 4.84713\times10^{-12}, 2.08387\times10^{-11},$$
$$8.59906\times10^{-11}, 3.40484\times10^{-10}, 1.29321\times10^{-9}, 4.71004\times10^{-9}, 1.64437\times10^{-8}, 5.50079\times10^{-8},$$
$$1.76246\times10^{-7}, 5.40608\times10^{-7}, 1.58675\times10^{-6}, 4.45421\times10^{-6}, 0.0000119516, 0.0000306352,$$
$$0.0000749673, 0.000175018, 0.000389525, 0.000825819, 0.00166633, 0.00319717,$$
$$0.0058273, 0.0100787, 0.0165222, 0.0256401, 0.037615, 0.0520891, 0.0679786, 0.0834569,$$
$$0.0961979, 0.103883, 0.104846, 0.0986361, 0.0862369, 0.0698336, 0.0521786, 0.0358168,$$
$$0.0224732, 0.0128143, 0.00659461, 0.00303798, 0.00124038, 0.000443352, 0.000136572,$$
$$0.0000355204, 7.58406\times10^{-6}, 1.27627\times10^{-6}, 1.58711\times10^{-7}, 1.29652\times10^{-8}, 5.21867\times10^{-10}\}$$

These probabilities are in the following order: p[1,K] = Pr (K-1 collisions), p[2,K] = Pr (K-2 collisions),..., p[K,K] = Pr (0 collisions). I want these probabilities in the reverse order, so I "reverse" the list above.

In[8]:=
```
rpr = Reverse[pr];
```

Now using these probabilities, I want to compute cumulative probabilities, and the number of collisions that correspond to the following percentages:

In[9]:=
```
pin = {.01, .05, .25, .5, .75, .95, .99, 1.};
```

The following loop computes the cumulative probabilities, and outputs the percentage points.

In[10]:=
```
s = 0;
j = 1;
Do[
 While[s < pin[[i]], s = s + rpr[[j]]; j++];
 Print[{s, j - 1}];
 If[s > 0.9999, Break[]],
 {i, 1, 8}]
```

```
{0.0114974, 11}

{0.0826017, 14}

{0.290851, 17}

{0.598216, 20}

{0.777871, 22}

{0.961194, 26}

{0.993622, 29}

{1., 52}
```

So this means, there are 11 or less collisions with probability 0.011. There are 26 or less collisions with probability 0.96.

An analytic formula

There is actually an analytic formula to compute the probability of, say, c collisions. The formula is based on a combinatorial result and uses Stirling numbers. The following code implements the formula. Per[c] below is the probability of having c collisions when K balls are tossed into M urns. This formula is computationally feasible for small values. I will use it to check the answer I got from the recursion.

In[13]:=
```
per[c_] :=
  Apply[Times, Table[M - i, {i, 0, K - c - 1}]] * StirlingS2[K, K - c] / M^K // N;
```

pra is the list of the probabilities.

In[14]:=
```
pra = Table[per[i], {i, 0, K - 1}];
```

If I printed this list, we would see that it equals the probabilities pr we got from the recursion. Instead of printing lots of numbers, I compute the cumulative probabilities for pra:

In[15]:=
```
s = 0;
j = 1;
Do[
 While[s < pin[[i]], s = s + pra[[j]]; j++];
 Print[{s, j - 1}];
 If[s > 0.9999, Break[]],
 {i, 1, 8}]
```

```
{0.0114974, 11}

{0.0826017, 14}

{0.290851, 17}

{0.598216, 20}

{0.777871, 22}

{0.961194, 26}

{0.993622, 29}

{1., 52}
```

Note that these cumulative probabilities are the same as before.

Testing random numbers

I will use the same example mentioned in the notes: we want to test 200 numbers. We consider the first 3 digits of these numbers. There are 1000 possible values (each digit takes 10 values 0 through 9). The following is all these 1000 values: they are the urns.

```
urns = Tuples[{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, 3];
```

Here is the 10th element of this urns list:

```
urns[[10]]

{0, 0, 9}
```

If you generate the random number 0.00923, then it will fall into this urn, since its first three digits (after the decimal points) are 0,0,9. If you generate another random number falling into the same urn, then you have a collision. To keep track of collisions, I create a list of 1000 zeros:

```
coll = ConstantArray[0, 1000];
```

The following loop generates a random number using *Mathematica*'s generator. Then it finds the urn it falls into, by looking at its first 3 digits. If the number falls into the 10th urn (as listed in urns), then it adds 1 to the 10th component of the above array coll. This procedure is repeated 200 times.

```
Do[
 r = Random[];
 dig = RealDigits[r, 10, 3, -1][[1]];
 pos = Flatten[Position[urns, dig], 1][[1]];
 val = coll[[pos]];
 coll = ReplacePart[coll, pos → val + 1],
 {200}]
```

The array coll contains a bunch of zeros and positive integers, 1000 of them, to be exact. If the 10th entry of coll is, say, 3, that means there were 2 collisions at the 10th urn. So you have to subtract 1 from positive entries (if it is 0, you do not subtract one), and then add all the numbers to get the total number of collisions in all urns. This is what I do next:

```
ncoll = Apply[Plus, Select[coll, # > 0 &] - 1]

22
```

Is 22 small, large? Looking at the cumulative probabilities we see that 22 is the 78% point, i.e., there is a 78% percent of chance of having 22 or less collisions. It looks good! No reason to doubt the randomness of the 200 numbers using this test.