Kernel: SageMath 10.7

Probabilistic Simulation

MATH 242 Modern Computational Math

Sometimes we want to answer questions of the form "what is the probability that...?" or "what is the average number of...?" These questions may have an exact (that is, theoretical) answer, but this answer might be very hard to find. However, we can often use computational simulation to approximate the answer to such questions.

We will begin our study of probabilistic simulation today with some simple problems, before considering a more complicated problem in the next few class sessions.

First, we need some import statements:

In [1]:
```
random()
```

Out[1]:
```
0.9622976993297814
```

Simulating coin flips

Suppose an unfair coin lands on heads with probability p, where $0 < p < 1$. First, we will use a random number generator to simulate such a coin flip and determine whether it lands on heads.

In [6]:
```
# simulating one coin flip
p = 0.7
r = random()
print(r)
result = r < 0.7
print(result)
```

Out[6]:
```
0.4162060035615126
True
```

Now simulate k flips and determine the number of heads that occur.

In [11]:
```
# simulating the number of heads that occur in k flips of a
coin that lands on heads with probability p
```

```
k = 100
p = 0.7
result = [ random() < p for i in range(k) ]
print(result)
sum(result)
```

Out[11]:
```
[True, True, True, True, True, True, False, False, True, True,
True, False, False, True, True, True, False, True, True, True,
True, True, False, True, True, True, True, False, True, True,
True, True, True, True, True, True, True, False, True, True, False,
True, True, True, True, True, True, False, True, True, False,
True, True, True, True, True, True, True, True, True, True,
False, False, True, False, True, False, True, True, True,
True, True, False, True, True, True, False, True, False, True,
True, True, True, True, True, True, True, True, True, True,
True, True, True, True, True, True, True, True, True, True,
False]
```

```
81
```

Write a function that returns the number of heads that result from k flips of a coin that lands on heads with probability p.

In [18]:
```
# function that returns the number of heads in k simulated
coin flips of a coin that lands on heads with probability p
def coinFlips(p, k):
    result = [ random() < p for i in range(k) ]
    # print(result)
    return sum(result)
```

In [19]:
```
coinFlips(0.7, 100)
```

Out[19]:
```
82
```

Now write a function that returns the proportion of heads that result from k flips of a coin that lands on heads with probability p.

In [20]:
```
# function that returns the proportion of heads in k
simulated flips of a coin that lands on heads with
probability p
def proportionHeads(p, k):
    return coinFlips(p, k)/k
```

In [22]:
```
proportionHeads(0.7, 10000).n()
```

Out[22]:
```
0.698900000000000
```

The Law of Large Numbers

In the context of coin flips, the **law of large numbers** says that as the number of coin flips increases, the sample proportion of heads converges to the theoretical proportion (p) of heads.

In [26]:
```
p=0.2
print( proportionHeads(p, 10).n() )
print( proportionHeads(p, 100).n() )
print( proportionHeads(p, 1000).n() )
print( proportionHeads(p, 10000).n() )
print( proportionHeads(p, 100000).n() )
print( proportionHeads(p, 1000000).n() )
```

Out[26]:
```
0.300000000000000
0.140000000000000
0.204000000000000
0.201000000000000
0.199940000000000
0.200406000000000
```

Central Limit Theorem

In the context of coin flips, the **central limit theorem** says that as the number of coin flips increases, the *distribution* of the sample proportion becomes normally distributed. The mean of the distribution is the theoretical proportion p, and the variance decreases as the number of coin flips increases.

Let's observe the central limit theorem by plotting the distribution of the proportion of heads in k flips of our unfair coin.

In [30]:
```
# compute N sample proportions, each of k coin flips
N = 10000
p = 0.7
k = 100
samples = [proportionHeads(p, k) for i in range(N)]

# make a histogram of the proportions
numBins = round((max(samples) - min(samples))*k + 1)
plotMin = min(samples) - 1/(2*k)
plotMax = max(samples) + 1/(2*k)
H = histogram(samples, density=True, color='c',
```

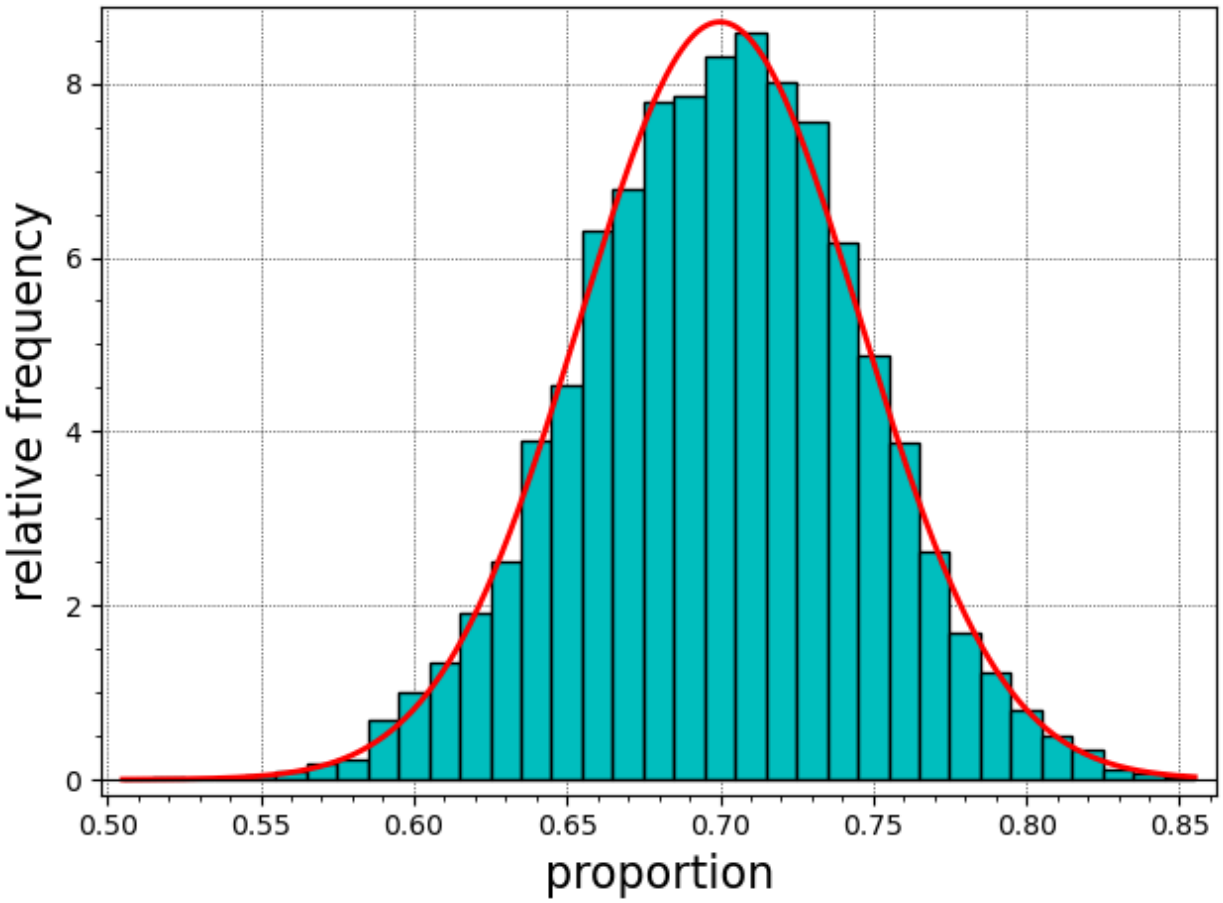
```
edgecolor='k', range=[plotMin, plotMax], bins=numBins,
axes_labels=["proportion","relative frequency"],
frame=True, gridlines=True, zorder=2)

# display normal distribution
normpdf(x, mu, sigma) = 1/(sqrt(2*pi)*sigma)*exp(-(x -
mu)^2/(2*sigma^2))

sigma = math.sqrt(p*(1-p)/k)
P = plot(normpdf(x, p, sigma), (x, plotMin, plotMax),
color='red', thickness=2)

show(H + P)
```

Out[30]:



Summary of common random number functions in Sage

The following examples demonstrate random number functions that you might want to use in class or on the homework.

In [31]:

```
# random decimal number between 0 and 1: random()
random()
```

Out[31]: 0.9761602458336965

In [32]:
```
# random number from a range: randrange(min, max, step)
randrange(7)
```

Out[32]: 5

In [34]:
```
# random selection from a list: choice(list)
testList = [2, 3, 5,7, 11]
choice(testList)
```

Out[34]: 7

The Birthday Problem

Here is a classic probability problem that we can approach via simulation.

Suppose that each person's birthday is chosen at random from 365 days in a year. (Ignore leap years, and suppose that birthdays are uniformly distributed.) How many people do you need so that the probability that two people share the same birthday is more than 0.5?

Here is one way to do it. This is certainly not the only way to structure the code.

In [1]:
```
# generate k birthdays
def generateBirthdays(k):
    return [randrange(365) for i in range(k)]
```

In [2]:
```
print( generateBirthdays(20) )
```

Out[2]: [202, 307, 186, 189, 241, 196, 28, 355, 176, 251, 78, 151, 132, 147, 2, 359, 355, 53, 148, 112]

In [3]:
```
# determine whether a list of birthdays contains a
duplicate birthday
def hasDuplicate(birthdayList):
    # loop over days
    for i in range(365):
        # check if this birthday occurs more than once
        if birthdayList.count(i) > 1:
            # testing: print(f"birthday {i} occurs
{birthdayList.count(i)} times")
            return True
```

```
    # if we get here, then there is no duplicate birthday
    return False
```

In [4]:
```
# testing
bdays = generateBirthdays(40)
print(bdays)
hasDuplicate(bdays)
```

Out[4]:
```
[266, 106, 234, 52, 358, 52, 25, 358, 96, 297, 12, 90, 56,
270, 8, 231, 67, 51, 10, 359, 6, 68, 254, 200, 240, 53, 347,
308, 36, 134, 1, 347, 327, 63, 248, 301, 5, 78, 214, 144]

True
```

In [7]:
```
# estimate the probability of a duplicate birthday
def probDuplicate(numPeople, numTrials):
    # perform numTrials trials, each finding whether a
group of numPeople people contains a duplicate birthday
    results = [hasDuplicate(generateBirthdays(numPeople))
for i in range(numTrials)]
    # return the proportion of the trials that resulted in
a duplicate birthday
    return sum(results)/numTrials
```

In [8]:
```
# testing
probDuplicate(20, 10).n()
```

Out[8]:
```
0.200000000000000
```

In [9]:
```
# now find the probability of a duplicate birthday for
various numbers of people
numTrials = 10000
for numPeople in range(10, 30):
    prob = probDuplicate(numPeople, numTrials)
    print(f"for {numPeople} people, the probability of a
duplicate birthday is {prob.n(digits=4)}")
```

Out[9]:
```
for 10 people, the probability of a duplicate birthday is
0.1168
for 11 people, the probability of a duplicate birthday is
0.1395
for 12 people, the probability of a duplicate birthday is
0.1670
for 13 people, the probability of a duplicate birthday is
0.1983
for 14 people, the probability of a duplicate birthday is
0.2248
for 15 people, the probability of a duplicate birthday is
```

```
0.2543
for 16 people, the probability of a duplicate birthday is
0.2844
for 17 people, the probability of a duplicate birthday is
0.3234
for 18 people, the probability of a duplicate birthday is
0.3424
for 19 people, the probability of a duplicate birthday is
0.3822
for 20 people, the probability of a duplicate birthday is
0.4136
for 21 people, the probability of a duplicate birthday is
0.4493
for 22 people, the probability of a duplicate birthday is
0.4711
for 23 people, the probability of a duplicate birthday is
0.5102
for 24 people, the probability of a duplicate birthday is
0.5381
for 25 people, the probability of a duplicate birthday is
0.5649
for 26 people, the probability of a duplicate birthday is
0.6047
for 27 people, the probability of a duplicate birthday is
0.6319
for 28 people, the probability of a duplicate birthday is
0.6571
for 29 people, the probability of a duplicate birthday is
0.6791
```

It looks like we need at least 23 people so that the probability of a duplicate birthday is at least 0.5.

If you found the answer to the birthday problem above, then consider one or more of the following questions.

Given N people:

- What is the probability that at least K people share the same birthday?
- What is the probability that exactly K people share the same birthday?
- What is the probability that two people share consecutive birthdays?
- What is the probability that K people have birthdays within Δ days of each other.

To make these problems more concrete, you may choose specific values for parameters such as N and K.

In [0]:

In [0]:

In [0]: