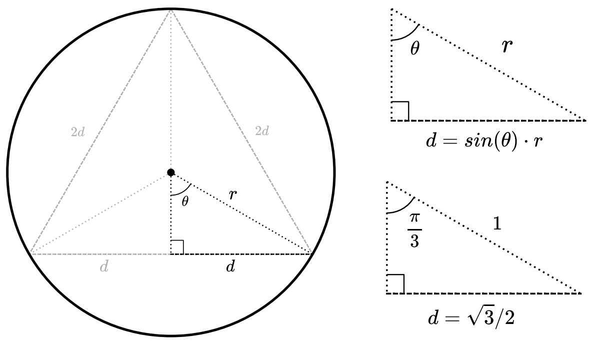
ICMA393 Discrete Simulation: HW2

Mahidol University International College
Austin Jetrin Maddison
October 10, 2024

1. Archimedes Method of N-Gon

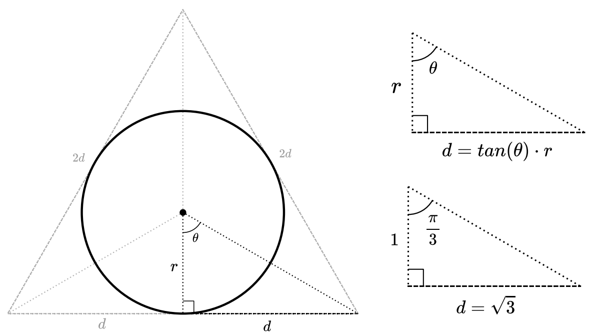
a)

Calculate p_3



$$
\begin{aligned}
p_3 &= 2d + 2d + 2d \\
&= 6d \\
&= 6 \cdot \frac{\sqrt{3}}{2} \\
&= 3\sqrt{3}
\end{aligned}
$$

Calculate P_3



$$
\begin{aligned}
P_3 &= 2d + 2d + 2d \\
&= 6d \\
&= 6 \cdot \sqrt{3} \\
&= 6\sqrt{3}
\end{aligned}
$$

b)

```
P_6 , p_6  pi approx = 3.14614428
pi                   = 3.14159265
# matching digits    = 3

P_12, p_12 pi approx = 3.14187328
pi                   = 3.14159265
# matching digits    = 4

P_24, p_24 pi approx = 3.14161018
pi                   = 3.14159265
# matching digits    = 4
```

```
P_48, p_48 pi approx = 3.14159375
pi                   = 3.14159265
# matching digits    = 6

P_96, p_96 pi approx = 3.14159272
pi                   = 3.14159265
# matching digits    = 7

P_192, p_192 pi approx = 3.14159266
pi                     = 3.14159265
# matching digits      = 9
```

2. Approximate $\sqrt{2}$

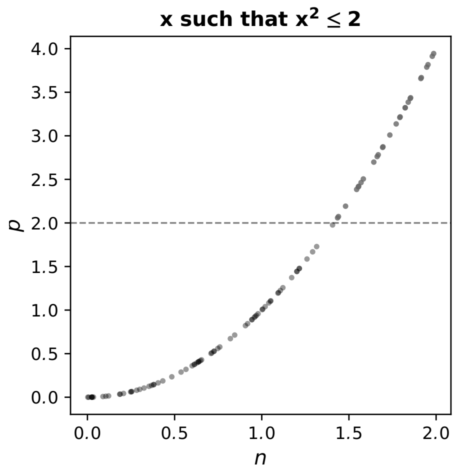
a) Summed all the n-terms of the taylor expansion of $\sqrt{2}$.

```
def f(n=150):
    x = 2
    sum = 0
    an = 1
    bn = 1/2

    for i in range(n):
        term = (an * (x - 1) ** bn) * (x - 1) ** i / math.factorial(i)
        sum += term

        an *= bn
        bn -= 1

    return sum
```

$f() = 1.4142909169379279$

b) Approximate $\sqrt{2}$ by uniformly sampling uniform random numbers x and counting whether x^2.

```
def f(n = 1000000, x=2):
    xs = np.random.ranf(n) * x
    ys = xs**2
    count = ys <= x
    res = np.mean(count) * x
    std = np.std(count) * x

    calc_ci = lambda x, std, n, z = 1.96 : [x -
        z * (std / np.sqrt(n)), x + z * (std /
        np.sqrt(n))]
    return res, calc_ci(res, std, n)
```



$f() = 1.4142909169379279$
$CI = [1.4135070094100637, 1.4170729905899362]$

3. Generalized Monty Hall

a)

```
def f(N, max_doors=3, max_cars=1, switch=True):
    max_cars = min(max_cars, max_doors)
    correct = np.zeros(N, dtype=bool)
    doors = np.arange(max_doors)

    for i in range(N):
    cars = np.random.choice(doors, size=max_cars, replace=False)
    my_choice = [np.random.randint(0, max_doors)]
    available_doors = np.setdiff1d(doors, np.union1d(cars, my_choice))

    host_open = np.random.choice(available_doors, size=1, replace=False)

    if switch:
        remaining_doors = np.setdiff1d(doors, host_open )
        remaining_doors = np.setdiff1d(remaining_doors, my_choice)
        my_choice = np.random.choice(remaining_doors, size=1, replace=False)

    correct[i] = my_choice in cars

    return np.mean(correct)
```

f(500000, max_doors=5, max_cars=2)
= 0.5333

b) I am going to count events like Ajarn demoed in class using a table of events as it was more intuitive to me:

	gggcc	ccggg	gccgg	gcggc	ggcgc	cgggc	ggccg	gcgcg	cggcg	cgcgg
Door 1	0	1	0	0	0	1	0	0	1	1
Door 2	0	1	1	1	0	0	0	1	0	0
Door 3	0	0	1	0	1	0	1	0	0	1
Door 4	1	0	0	0	0	0	1	1	1	0
Door 5	1	0	0	1	1	1	0	0	0	0

$$P\{\text{Chose C first}\} = \frac{20}{50} = \frac{2}{5} \quad P\{\text{Chose G first}\} = \frac{30}{50} = \frac{3}{5}$$

	cgg	gcg	ggc
Door 1	0	1	0
Door 2	1	0	0
Door 3	0	0	1

	ccg	gcc	cgc
Door 1	1	0	1
Door 2	1	1	0
Door 3	0	1	1

$$P\{\text{Winning | Chose C First}\} = \frac{3}{9} = \frac{1}{3} \qquad P\{\text{Winning | Chose G First}\} = \frac{6}{9} = \frac{2}{3}$$

$$P\{\text{Winning}\} = \frac{2}{5} \cdot \frac{1}{3} + \frac{3}{5} \cdot \frac{2}{3} = 0.5333\ldots$$

4. Truncated Arctan Series From Class

I did not do TT

5. Yet another way to approximate π

Inside `source/p5.ipynb`

6. Newton's Method

Inside `source/p6.ipynb`

Source Code

https://github.com/AustinMaddison/discrete-simulation/tree/main/hw2/source