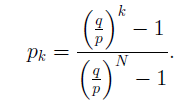
1. Теоретическая часть

Классическая задача о разорении игрока рассматривалась и была решена в XIX веке. Постановка задачи такова: пусть двое игроков играют в некоторую игру, причем первый выигрывает каждую партию независимо от других с вероятностью p, проигрывает с вероятностью q = 1 − p, а ничьих не бывает. В каждой партии разыгрывается ставка в 1 рубль. Первоначальный капитал первого игрока составляет k рублей, а второго - N − k рублей. Партии продолжаются до полного разорения одного из игроков, то есть до того момента, когда ему будет нечего поставить на кон. Можно представить условие задачи как случайное блуждание некоторой фигурки на отрезке прямой: если капитал первого игрока на текущий момент равен x, то фигурка находится в точке в координатой x. После следующей партии она смещается либо в x + 1, либо вx−1, в зависимости от того, выиграл или проиграл первый игрок эту партию. Если фигурка попадает в точку с координатой 0 или N, случайное блуждание заканчивается - происходит разорение одного из игроков. Задача имеет аналитическое решение. Если, как было указано, первоначальный капитал первого игрока составляет k рублей, а второго - N − k рублей, то вероятность выигрыша первого игрока, то есть разорения второго, равна



1. Программный код

"""

Программа, производящая вычисление по формуле

вероятности общей победы и строящая графики зависимости

в задаче о разорении игрока

Автор: Афанасьев И.Е.

Дата написания: 20.09.2020

"""

# Функция расчёта вероятности общей победы

def fun(p, k, N):

q = 1 - p

return ((q/p)\*\*k - 1) / ((q/p)\*\*N - 1)

# импорт графической библиотеки

import matplotlib.pyplot as plt

# Задаём пул различных параметров задачи

# P - массив вероятность выигрыша первого игрока

# N - массив общего количества денег игроков

# K - массив количества денег первого игрока (n - k - деньги второго игрока)

P = [i/100 for i in range(55, 100, 5)]

N = [i for i in range(5, 15, 1)]

K = [i for i in range(1, 10, 1)]

# для первого задания

# Фиксируем p и k, n - меняем

funcN = []

for n in N:

funcN += [fun(P[len(P) // 2], K[len(K) // 2], n)]

# Фиксируем n и k, p - меняем

funcP = []

for p in P:

funcP += [fun(p, K[len(K) // 2], N[len(N) // 2])]

# Фиксируем p и n, k - меняем

funcK = []

for k in K:

funcK += [fun(P[len(P) // 2], k, N[len(N) // 2])]

# Строим графики

fig = plt.figure()

plt.subplot(4, 1, 1)

plt.plot(P, funcP)

plt.subplot(4, 1, 2)

plt.plot(K, funcK)

plt.subplot(4, 1, 3)

plt.plot(N, funcN)

# для второго задания

# k и n меняются в соотношении 1 к 10

K = [i for i in range(1,6,1)]

N = [i for i in range(10,60,10)]

p = 0.55

# Вычисляем по формуле

funcNK10 = []

for i in range(len(K)):

funcNK10 += [fun(p, K[i], N[i])]

# Строим графики

plt.subplot(4, 1, 4)

plt.plot(K, funcNK10)

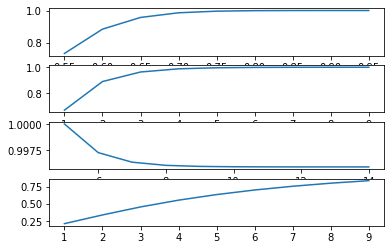
# Выводим на экран

plt.show()

1. Результаты работы программы

Графики:

1. Зависимость вероятности общей победы от p – вероятности победы первого игрока
2. Зависимость вероятности общей победы от k – числа денег первого игрока в начале
3. Зависимость вероятности общей победы от n – общего числа денег
4. Зависимость общей победы от k при соотношении k / n = 0.1



Чтобы первый игрок почти наверняка выигрывал: вероятность p -> 1

Умение играть важнее

Теоретические вопросы:

1. При p -> 0 – pk -> 0

При p -> 1 – pk -> 1

Эти пределы зависят от N и k только при условии N = k (У второго игрока денег нет изначально, тогда при любом p – первый игрок уже выиграл)

Или при условии k = 0 (Аналогично, второй игрок уже выиграл)

При N = 0 – решения нет, поскольку не имеет смысла, игра вообще не состоится (капитала нет изначально)

1. Pk = k! / (N \* (N – 1) \* (N – 2) \* … \*(N – k + 1))
2. Более богатого.