Статистика, прикладной поток

Практическое задание

В данном задании вы изучите свойства метода Монте-Карло, сравнив его с методом прямоугольков. Также с помощью метода Монте-Карло вы решите задачу, которая возникает при составлении проекта по разработке нефтяного месторождения. Знания физики или экономики не требуются.

Правила:

- Дедлайн **26 сентября 23:59**. После дедлайна работы не принимаются кроме случаев наличия уважительной причины.
- Выполненную работу нужно отправить на почту `mipt.stats@yandex.ru`, указав тему письма "`[applied] Фамилия Имя - задание 1`". Квадратные скобки обязательны.
- Прислать нужно ноутбук и его pdf-версию (без архивов). Названия файлов должны быть такими: `1.N.ipynb` и `1.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- Решения, размещенные на каких-либо интернет-ресурсах не принимаются. Кроме того, публикация решения в открытом доступе может быть приравнена к предоставлении возможности списать.
- Для выполнения задания используйте этот ноутбук в качестве основы, ничего не удаляя из него.
- Никакой код из данного задания при проверке запускаться не будет.

Баллы за задание:

- Задача 1 - 1 балл
- Задача 2 - 10 баллов
- Задача 3 - 10 баллов
- Задача 4 - 5 баллов
- Задача 5 - 20 баллов

Все задачи имеют тип **О3**. Подробнее см. в правилах выставления оценки.

Задача 1. Найдите книгу Савельев В. "Статистика и котики" и прочитайте главы 1 и 2. Какие выводы можно сделать?

Вывод: <...>

Задача 2. Реализуйте метод Монте-Карло и метод прямоугольников численного интегрирования функции. Реализация должна уметь вычислять интеграл вида:

$$\int\limits_{l_1}^{h_1} \dots \int\limits_{l_d}^{h_d} f(x_1, \dots, x_d) dx_1 \dots dx_d$$

Детали реализации: на вход функции подаются векторы $l = (l_1, \ldots, l_d)$ и $h = (h_1, \ldots, h_d)$, число n -- максимальное допустимое число вызовов функции f (если вы не делаете лишних вызовов, оно равно числу точек-центров отрезков, прямоугольников, параллелипипедов, и т.д. в многомерных случаях). Использование циклов, кроме циклов по числу интегралов d, **наказуемо**. Используйте функции `numpy.meshgrid` и `numpy.vectorize` для быстрой скорости работы.

In [1]:

```
import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt
import time

%matplotlib inline
```

Пример использования `numpy.vectorize` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.vectorize.html).

In [79]:

```
def vector_function(x):
    """Получает на вход вектор некоторой длины n, возвращает число.
    Сигнатуру можно записать как (n)->()"""
    return x.sum()

f = np.vectorize(vector_function, signature="(n)->()")
arg = np.arange(10).reshape(-1, 2)
arg
```

Out[79]:

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

In [80]:

```
f(arg)  # вычисляет суммы по строкам
```

Out[80]:

```
array([ 1,  5,  9, 13, 17])
```

Пример использования `numpy.meshgrid` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.meshgrid.html). Как всегда, настоятельно советуем читать документацию.

In [81]:

```
# координаты точек на некоторой координатной оси
linspace_x = [1, 2, 3]
linspace_y = [-1, -2, -3]

# возвращает матрицы с координатами сетки,
# задаваемой проекциями точек на оси координат
X, Y = np.meshgrid(linspace_x, linspace_y)

# преобразуем исходный формат к вектору точек
np.stack([X, Y]).reshape(2, -1).T
```

Out[81]:

```
array([[ 1, -1],
       [ 2, -1],
       [ 3, -1],
       [ 1, -2],
       [ 2, -2],
       [ 3, -2],
       [ 1, -3],
       [ 2, -3],
       [ 3, -3]])
```

In [86]:

```
class Integrator:
    @staticmethod
    def integrate(f, low, high, n, method="rectangle"):
        """Вычисление интеграла.
        f - функция многих переменных, на вход принимает вектор;
        low - нижние границы (в том же порядке, в котором
                              функция принимает аргументы);
        high - верхние границы (аналогично);
        n - максимальное число вызовов функции f;
        method - метод ("rectangle" или "monte_carlo",
                            см. Integrator.methods);
        """

        assert len(low) == len(high)
        low, high = map(np.array, [low, high])
        n = int(n)

        return Integrator.methods[method](f, low, high, n)

    def integrate_monte_carlo(f, low, high, n):
        """Метод монте-карло"""

        # случайные точки, в которых будем вычислять функцию
        dots = np.random.uniform(low=low, high=high, size=(n, len(low)))
        # вычисление функции в случайных точках
        return np.sum(f(dots)) * np.prod(high-low)/n

    def integrate_rectangle(f, low, high, n):
        """Метод прямоугольников"""

        #размерность
        dim = len(low)
        # число точек для каждой координаты
        n_for_one_dim = int(np.power(n, 1./dim))
        # разбиения отрезков интегрирования на равные отрезки
        ranges = np.array([np.linspace(low[i], high[i], n_for_one_dim+1) for i in range
(dim)])
        # вычисление центров этих разбиений
        ranges_of_centers = (ranges[:,:-1]+ranges[:,1:])/2
        # получение всех точек сетки через вызов np.meshgrid
        centers = np.stack(np.meshgrid(*ranges_of_centers)).reshape(dim, -1).T
        # длины длины отрезков по каждой координате
        block_lengths = (high-low)/n_for_one_dim
        # вычисление значение функции в точках сетки
        f_values = f(centers)
        # ответ
        ans = np.sum(f_values) * np.prod(block_lengths)
        return ans

    methods = {
        "rectangle": integrate_rectangle,
        "monte_carlo": integrate_monte_carlo
    }
```

Вычислите $\int\limits_0^1 \int\limits_3^4 \left(x^2 + y^2\right) dxdy$ на миллионе запусков функции $f(x, y) = x^2 + y^2$ двумя рассмотренными методами. Измерьте время работы методов и сравните результат с истинными значением интеграла. Различается ли время работы методов?

In [3]:

```
def time_and_accuracy():
    low = np.array([0,3])
    high = np.array([1,4])
    n = 1000000
    theoretical_value = (4**3 - 3**3 + 1)/3.

    def f(x):
        return x[0]**2 + x[1]**2

    vector_f = np.vectorize(f, signature="(2)->()")

    methods = ["rectangle", "monte_carlo"]

    integrator = Integrator
    computation_time = {}
    result = {}

    for method in methods:
        start_time = time.time()
        result[method] = integrator.integrate(f=vector_f, low=low, high=high, n=n, meth
od=method)
        computation_time[method] = time.time()-start_time

    print("Время работы метода прямоугольников: ", computation_time["rectangle"], ", "
          "Монте-Карло: ",  computation_time["monte_carlo"])
    print("Разность рассчитанного и реального значений для прямоугольников: ", result[
"rectangle"] - theoretical_value,
          ", для Монте-Карло: ", result["monte_carlo"] - theoretical_value)

time_and_accuracy()
```

```
Время работы метода прямоугольников:  19.59290051460266 , Монте-Карло:  1
9.1209876537323
Разность рассчитанного и реального значений для прямоугольников:  -1.66666
66624587378e-07 , для Монте-Карло:  -0.0011777487007833543
```

Комментарий: Для этой задачи метод Монте-Карло и метод прямоугольников работают одинаково по времени. Но метод прямоугольников значительно точнее.

Задача 3. Для $d = 1...8$ оцените скорость сходимости методов для интеграла

$$\int\limits_0^1 \dots \int\limits_0^1 \sum_{i=1}^{d} x_i^2 \prod_{i=1}^{d} dx_i$$

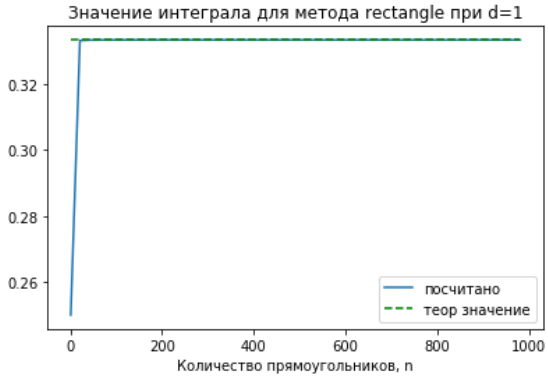
Т.е. $\int\limits_0^1 x_1^2 dx_1$, $\int\limits_0^1 \int\limits_0^1 \left(x_1^2 + x_2^2\right) dx_1 dx_2$ и так далее.

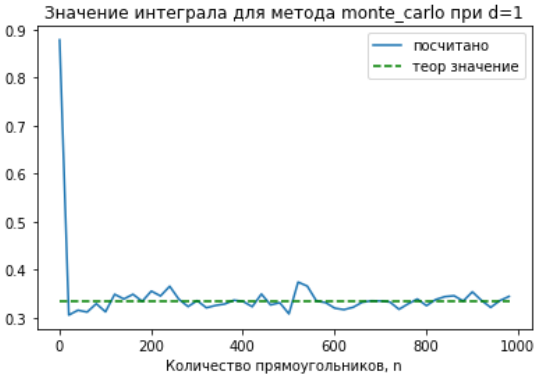
Вычислите точное значение этого интеграла и для каждого d постройте график зависимости вычисленного значения интеграла от числа $n = d \times 1000$ вызовов подынтегральной функции (в корректном решении равно числу точек-центров), которое разрешено использовать для каждого метода вычисления интеграла. На графике укажите точное значение интеграла. Для наглядности графики рекомендуется расположить в два столбика.

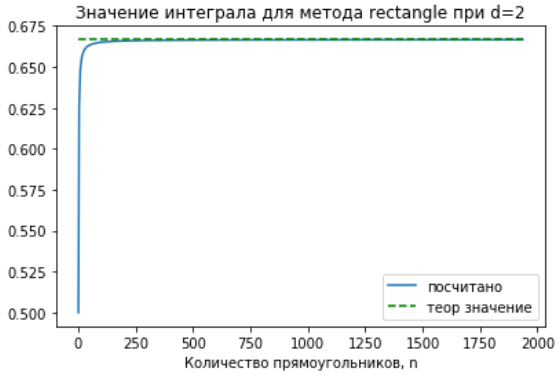
Какой метод и при каких d сходится быстрее? Предположите, в каком случае выгоднее использовать тот или иной метод.

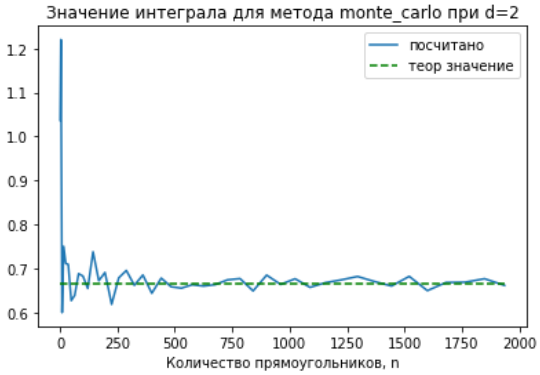
In [34]:

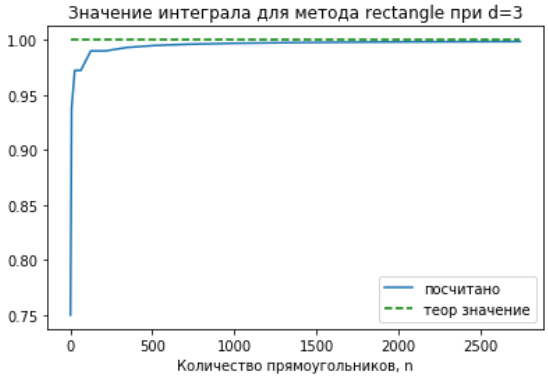
```
def plot_convergence(f, low, high, n_list, theoretical_value):
    """Строим графики сходимости.
    f - функция многих переменных, на вход принимает вектор;
    low - нижние границы (в том же порядке, в котором
                          функция принимает аргументы);
    high - верхние границы (аналогично);
    n_list - список;
    theoretical_value - точное значение интеграла.
    """

    methods = ["rectangle", "monte_carlo"]

    integrator = Integrator

    plt.figure(figsize=(14, 4))
    counter=1

    for method in methods:
        calculated_values = [integrator.integrate(f=f, low=low, high=high, n=n, method=
method) for n in n_list]

        plt.subplot(1, 2, counter)
        counter+=1

        plt.plot(n_list, calculated_values, label='посчитано')
        plt.plot([0, n_list[-1]], theoretical_value*np.ones(2), "g--", label='теор знач
ение')
        plt.title('Значение интеграла для метода '+str(method)+' при d='+str(len(low)))
        plt.xlabel(r'Количество прямоугольников, n')
        plt.legend()

    plt.show()



max_d = 8

for d in range(1, max_d+1):
    def f(x):
        return np.sum(np.square(x))

    f = np.vectorize(f, signature="(d)->()")
    n_list = np.power(range(1, 1 + int((d*1000)**(1./d))), d)
    if d==1:
        n_list = range(1, 1000, 20)

    plot_convergence(f, np.zeros(d), np.ones(d), n_list, d/3.)
```

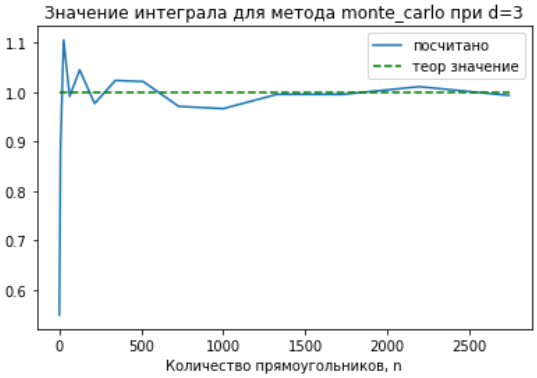

Значение интеграла для метода rectangle при d=1
посчитано
теор значение
Количество прямоугольников, n

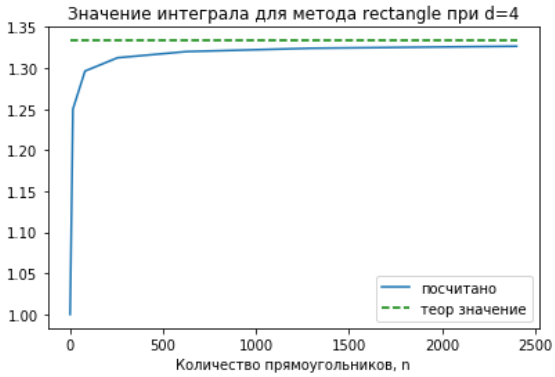

Значение интеграла для метода monte_carlo при d=1
посчитано
теор значение
Количество прямоугольников, n

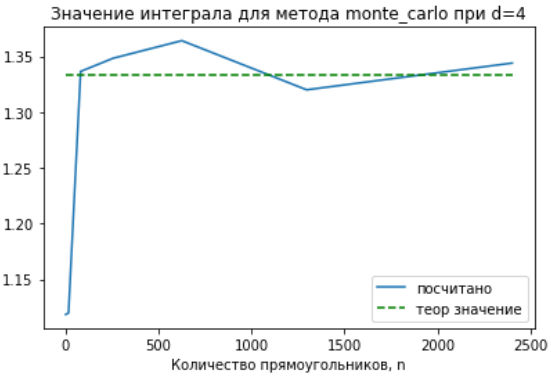

Значение интеграла для метода rectangle при d=2
посчитано
теор значение
Количество прямоугольников, n

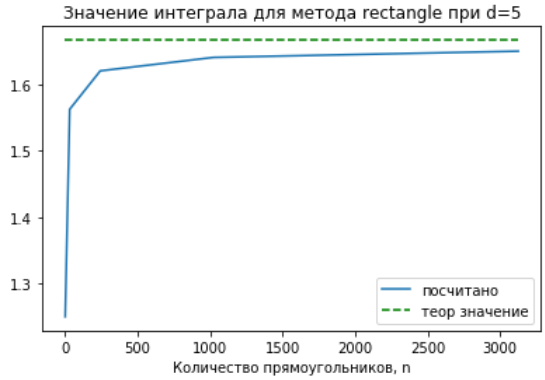

Значение интеграла для метода monte_carlo при d=2
посчитано
теор значение
Количество прямоугольников, n

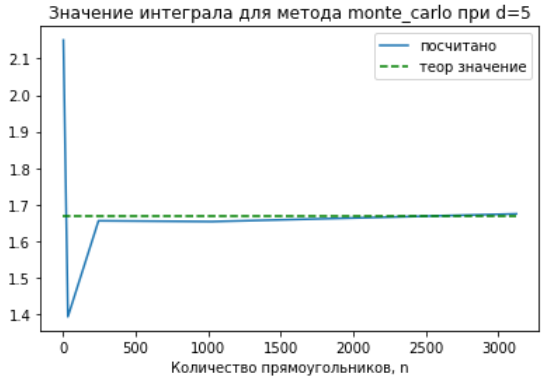

Значение интеграла для метода rectangle при d=3
посчитано
теор значение
Количество прямоугольников, n

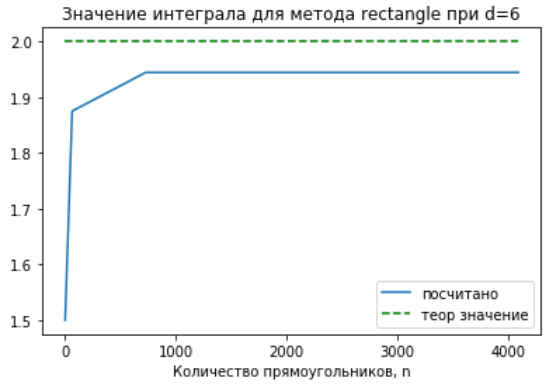

Значение интеграла для метода monte_carlo при d=3
посчитано
теор значение
Количество прямоугольников, n

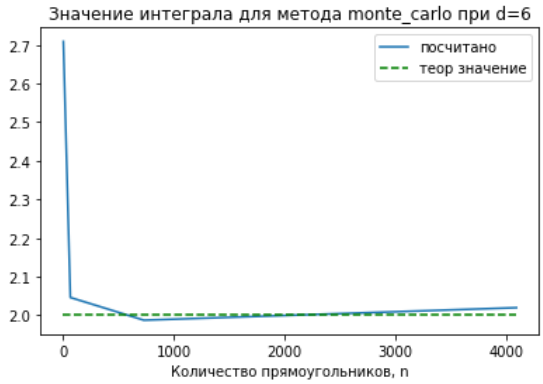

Значение интеграла для метода rectangle при d=4
посчитано
теор значение
Количество прямоугольников, n

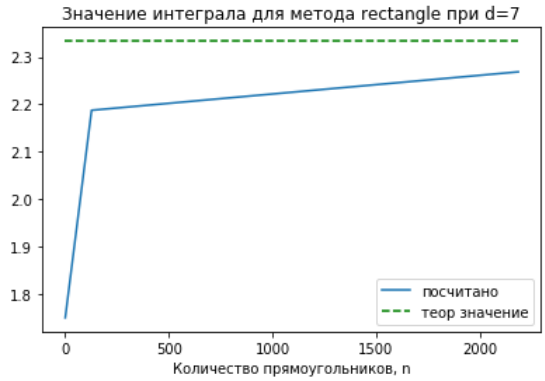

Значение интеграла для метода monte_carlo при d=4
посчитано
теор значение
Количество прямоугольников, n

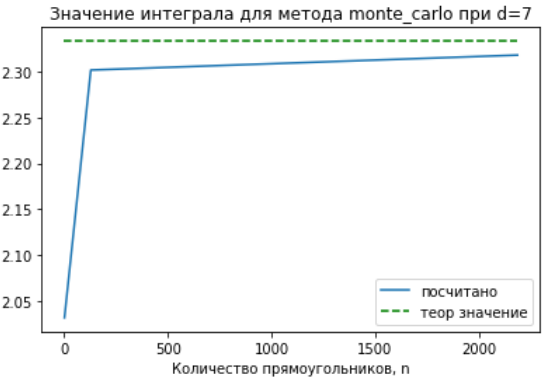


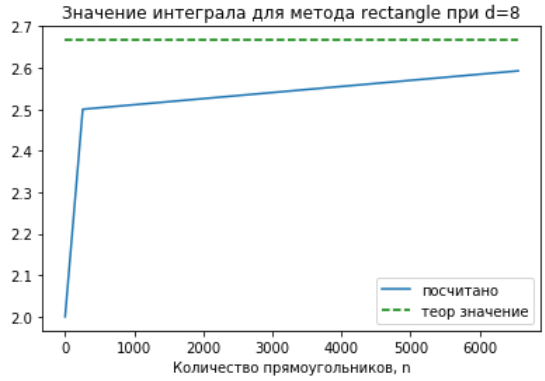


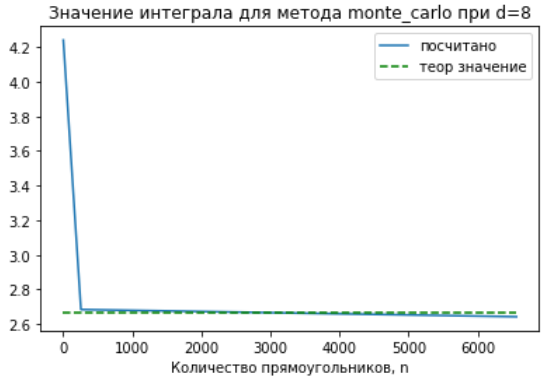












Вывод: Для больших d интеграл сходится быстрее при использовании метода Монте-Карло. При меньших размерностях сходится быстрее метод прямоугольников. Также метод прямоугольников накладывает большие ограничения на количество прямоугольников. Например, при d=8 возможны только 3 значения n, меньшие 8000. Таким образом, метод Монте-Карло удобнее использовать на больших размерностях.

Задача 4. Вам предлагается численно вычислить многомерный интеграл Пуассона для $d = 5$ и некоторой симметричной положительно определенной матрицы A, которую вы выберете сами. Зависит ли интеграл от выбора A?

$$\int\limits_{\mathbb{R}^n} exp(-x^T Ax)dx$$

Сравните результаты двух методов с истинным значением интеграла. Как вы думаете, какой метод выдает более точный результат? Количество итераций каждого метода должно быть не менее 10^6.

In [92]:

```
def integrate_Poisson(d, A, n, range):
    """Вычисляет интеграл Пуассона размерности d для матрицы A,
    n - количество вызовов подинтегральной функции
    range - верхняя граница интеграла (-range это нижняя)
    """
    integrator = Integrator

    def f(x):
        return np.exp(-np.matmul(np.matmul(x.T,A), x))


    f = np.vectorize(f, signature="(d)->()")

    high = np.ones(d)*range
    low = -high

    return (integrator.integrate(f, low, high, n),
            integrator.integrate(f, low, high, n, "monte_carlo")
    )

result = integrate_Poisson(5, np.eye(5), 1000000, 5)
print("Интеграл Пуассона для единичной матрицы. Значение интеграла методом прямоугольни
ков: ", result[0], ". ",
      "Значение посчитанное методом Монте-Карло: ", result[1])
```

```
Интеграл Пуассона для единичной матрицы. Значение интеграла методом прямоу
гольников:  17.493418367284338 .  Значение посчитанное методом Монте-Карл
о:  17.889181142032353
```

Комментарий: Интеграл зависит от выбора A, например, если взять $A = E/a$, то значение интеграла будет равно $(a\pi)^{2.5}$. Если взять $A = E$, то теоретическое значение интеграла будет равно 17,4934183276. Получается, что метод прямоугольников все еще точнее считает значение интеграла.

Рассмотрим отношение интегралов

$$F(t) = \frac{\int\limits_{-\infty}^{t_1} \dots \int\limits_{-\infty}^{t_k} exp\left(-\frac{1}{2}x^T Ax\right) dx}{\int\limits_{\mathbb{R}^n} exp\left(-\frac{1}{2}x^T Ax\right) dx}$$

В чем его вероятностный смысл?

Ответ: Это функция распределения гаусового вектора с нулевым матожданием и матрицей ковариаций A^{-1}.

Задача 5. В заключении рассмотрения метода Монте-Карло вам предлагается реальная практическая задача.

На основе http://ecsocman.hse.ru/data/819/759/1219/Monte_Karlo_dlya_analitikov.pdf (http://ecsocman.hse.ru/data/819/759/1219/Monte_Karlo_dlya_analitikov.pdf)

Рассмотрим проект по разработке нефтяного месторождения. В основе модели проекта лежат предварительные данные о величине резервов месторождения.

Замечание. Знания физики или экономики не требуются.

Формулировка задачи от заказчика: Общая задача анализа --- основываясь на величине запасов и проценте нефтеотдачи рассчитать NPV (чистая приведенная стоимость) проекта, а точнее, 0.1-квантиль ее распределения. Следующим этапом мы хотим использовать ее в качестве критерия оптимизации, то есть максимизировать такое значение NPV, которого мы можем достигнуть или превысить с 90%-й вероятностью, подобрав при этом оптимальное количество скважин на месторождении.

Предположим, что на месторождении есть 25 скважин. Эти скважины добывают некоторую смесь, которая состоит из воды, нефти и различных примесей. Доля нефти из добытого материала называется коэффициентом нефтеотдачи (https://ru.wikipedia.org/wiki/Нефтеотдача) k. Мы будем считать, что этот коэффициент является одинаковым для всего месторождения и имеет нормальное распределение со средним 42% и стандартным отклонением 1.2%.

Добыча нефти скважиной за год

Разработка месторождения (http://vseonefti.ru/upstream/stadii-razrabotki.html) включает три этапа:

1. фаза роста добычи --- период введения в работу новых скважин;
2. фаза плато: после достижения определенного уровня добычи, она некоторое время продолжается на постоянном уровне;
3. фаза снижения добычи --- период, когда темпы добычи экспоненциально снижаются с течением времени.

Для упрощения задачи мы пропустим два первых этапа и рассмотрим только последний.

Каждая скважина j характеризуется параметром q_j --- темп добычи из скважины, определяемый объемом вещества (смесь нефти, воды и др.), добываемого скважиной за сутки. Будем считать, что этот параметр является одинаковым для скважины в течении всего периода разработки и имеет нормальное распределение со средним 10 тыс. баррелей и стандартным отклонением 3 тыс. баррелей. Темпы добычи для разных скважин считаются независимыми случайными величинами.

Соответственно, за год t скважина добывает $Q_{tj} = 365 \cdot k \cdot q_j \cdot e^{-0.2(t-1)}$ тыс. баррелей нефти, где экспонента отвечает за снижение добычи с течением времени. Всего за год t на месторождении добывается $Q_t = \sum\limits_{j=1}^{25} Q_{tj}$ тыс. баррелей нефти.

Прибыль

Стоимость барреля нефти будем считать постоянной и равной $c = 70$ долларов за баррель. Однако, для расчета стоимости нужно учесть ставку дисконтирования (https://ru.wikipedia.org/wiki/Ставка_дисконтирования) --- процентная ставка, используемая для пересчета будущих потоков доходов в единую величину текущей стоимости (см. формулу далее). Обозначим ее i и будем считать, что она имеет нормальное распределение со средним 10% и стандартным отклонением 1.2%.

Стоимость добытой нефти за год t составит (тыс. баррелей)

$$\frac{c \cdot Q_t}{(1+i)^{t-1}}.$$

Будем считать, что разработка месторождения прекращается, если за год на всем месторождении было добыто менее 100 тыс. баррелей нефти. Последний год разработки обозначим T.

Затраты

Затраты на месторождение (кроме скважин) составляют $C_{\text{мест.}} = 200$ млн. долларов в год. Будем считать, что издержки на содержание скважины j за весь период разработки имеют треугольное распределение (scipy.stats.triang (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.triang.html)) с минимальным значением 90 млн. долларов, максимальным значением 120 млн. долларов и модой (наиболее вероятное значение) 100 млн. долларов. Обозначим эти случайные величины $C^j_{\text{скв.}}$ и будем считать их независимыми.

NPV

Теперь мы можем написать формулу NPV

$$NPV = \sum_{t=1}^{T} \frac{c \cdot Q_t}{(1+i)^{t-1}} - T \cdot C_{\text{мест.}} - \sum_{j=1}^{25} C^j_{\text{скв.}}.$$

Задание

С помощью метода Монте-Карло требуется найти число x, при котором

$$\mathsf{P}(NPV < x) = 0.1.$$

Количество итераций метода должно быть не менее $100\,000$. На основе проделанных итераций оцените также среднее значение NPV и вероятность, с которой NPV будет положительна. Кроме того, постройте нормированную гистограмму значений NPV с помощью `plt.hist(values, bins=200, normed=True)`.

Перечислим еще раз все *случайные параметры*:

- Коэффициент нефтеотдачи k имеет нормальное распределение со средним 42% и стандартным отклонением 1.2%;
- $q_1, \ldots, q_{25}$ --- темпы добычи из скважин --- независимые нормальные случайные величины со средним 10 тыс. баррелей и стандартным отклонением 3 тыс. баррелей;
- Ставка дисконтирования i имеет нормальное распределение со средним 10% и стандартным отклонением 1.2%;
- $C^1_{\text{скв.}}, \ldots, C^{25}_{\text{скв.}}$ --- затраты на каждую скважину --- независимые случайные величины, имеющие треугольное распределение с минимальным значением 90 млн. долларов, максимальным значением 120 млн. долларов и модой (наиболее вероятное значение) 100 млн. долларов. (используйте `sps.triang(loc=90, c=1/3, scale=30)`)

Фиксированные параметры:

- 25 скважин;
- 365 дней в году;
- $c = 70$ долларов за баррель --- стоимость нефти;
- 100 тыс. баррелей --- объем добытой нефти за год, при котором разработка месторождения прекращается.
- 200 млн. долларов в год --- затраты на месторождение.

In [ ]:

```
# место для вашего решения
```

Вывод: <...>