МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

Институт информационных технологий и компьютерных наук

Кафедра инженерной кибернетики

Курсовая работа

по дисциплине «Численные методы»

на тему:

«Метод Монте-Карло.

Применение метода для численного решения ДУ с частными производными и вычисления кратных интегралов»

Выполнили:

студенты 3-го курса,

гр. БПМ-21-3 Кочян Л. В.

гр. БПМ-21-3 Ким Л. О.

Проверил:

Ширкин С. В.

Москва 2023

Оглавление

Введение

Метод Монте-Карло является группой численных методов для изучения случайных процессов, ключевой особенность является использование случайных величин большого количества для нахождения определенной характеристики, которая может быть не точной, но приблизительно равна истине.

В рамках данной работы мы рассмотрим использование метода Монте-Карло для интегрирования кратных интегралов и нахождению численного решения дифференциальных уравнений в частных производных на языке Python.

Помимо этого, будет освещена краткая история возникновения метода и его польза на практике.

История появления

Использовать случайные величины для практических задач идея не новая, к примеру, одним из методов найти число Пи было бросание иголок определенной длины на ряд параллельных прямых, придуманный в конце 18-го века, но так как природа таких расчетов подразумевает приближенный результат, то использовать их, считая вручную огромное количество величин не предоставляется эффективным, посему развитие их пришлось на момент создания первых вычислительных комплексов.

Метод Монте-Карло придумал американский математик польского происхождения Станислав Улам. С 1943 года он работал в лаборатории Лос-Аламос, в Манхэттенском проекте, в 1946 году он заболел. Выздоравливая, он проводил время за пасьянсом Кэнфилд. В один момент он заинтересовался - какова вероятность того, что пасьянс сойдется. Так как комбинаторика не могла ему дать точный отчет, то решил он считать “напролом” большим числом испытаний. После своего эксперимента он решил применить его в ядерной физике, на тот момент была задача с процессов рассеивания нейтронов на ядрах Улам сообразил, что его подход даст решение сложного дифференциального уравнения, только его надо предоставить авторитетным ученым. Он поделился своими соображениями с Нейманом, и они решили опробовать идею на реальной задаче. В Лос-Аламосе соблюдали секретность, и задаче дали имя – Монте-Карло, с намеком на дядю Улама – азартного игрока.

Таким образом, Монте-Карло стал одной из составляющей успеха в создании водородной бомбы.

Прикладное применение

Монте-Карло применяется во многих сферах жизни, даже на лекциях по био-хемоинформатике первый метод, который изучают, это именно Монте-Карло, так как хемоинформатика напрямую связана с моделированием большого количества молекул, которые и выступают как случайные величины.

Известны случаи применения Монте-Карло для расчета стоимости опциона путем генерирования случайных траекторий цен.

Для краткости приведены случаи, где метод может быть полезен:

- моделирование облучения твёрдых тел ионами в физике;
- моделирование поведения разреженных газов
- исследования поведения разных тел при столкновении
- алгоритмы оптимизации и нахождения кратчайшего пути решения
- решение сложных интегралов (или когда их очень много)
- предсказание астрономических наблюдений
- поиск в дереве в различных алгоритмах
- алгоритмы работы некоторых функций квантового компьютера
- моделирование состояния приближённой физической среды

Теоретико-практическая часть

Вычисление кратных интегралосв

Для функции одной переменной

Если x_i принадлежит [a,b], где i = 0,1,…,n,то используя метод можно вычислить следующим образом:

$$\int_a^b f(x)dx = \frac{b-a}{n} * \sum_{i=0}^{N} f(x_i)$$

Рассмотрим на следующем примере

$$\int_{-2}^{1} \frac{1}{\sqrt{x+3} + \sqrt{(x+3)\hat{}2}} dx$$

Пошаговая инструкция расчета:

1. Определение функции
2. Определение нижней и верхней границ, а также число генерируемых рандомных чисел
3. Генерация рандомных чисел
4. Нахождение суммы
5. Вычисление интеграла по формуле

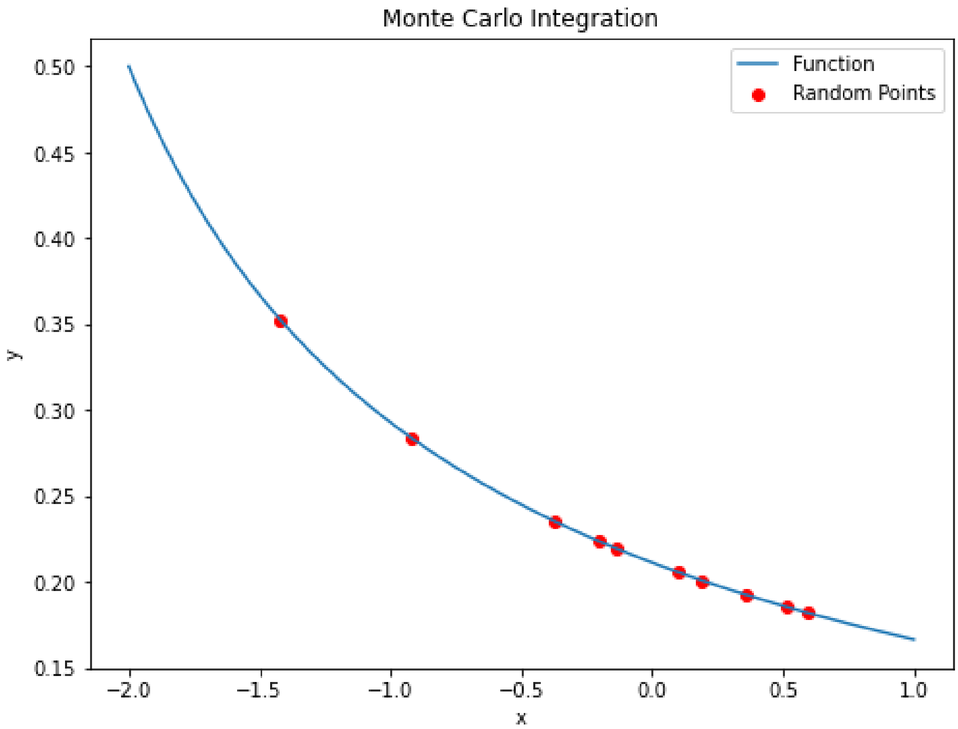
Код представлен на Питоне, большая часть кода — это расчёт графического представления результатов:

```
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.integrate import quad, nquad, quadrature, odeint
from typing import Tuple, Callable


class MonteCarloIntegration:
    def __init__(self, func, a, b, N):
        self.func = func
        self.a = a
        self.b = b
        self.N = N
        self.array = np.random.uniform(a, b, N)

    def calculate_integral(self):
        summa = np.sum(self.func(self.array))
        integration = (self.b - self.a) / self.N * summa
        return integration

    def calculate_exact_integral(self):
        result, _ = quad(self.func, self.a, self.b)
        return result

    def visualize_function(self):
        x_vals = np.linspace(self.a, self.b, 1000)
        y_vals = self.func(x_vals)

        plt.figure(figsize=(8, 6))
        plt.plot(x_vals, y_vals, label='Function')
        plt.scatter(self.array, self.func(self.array), color='red',
label='Random Points')
        plt.title('Monte Carlo Integration')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend()
        plt.show()

if __name__ == "__main__":

    def custom_function(x):
        return 1 / (np.sqrt(x + 3) + np.sqrt((x + 3)**2))

    a = float(input("Введите нижний предел: "))
    b = float(input("Введите верхний предел: "))

    N_s = [10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000]

    for N in N_s:

        print("Количество рандомных чисел:", N)
```

```
        monte_carlo = MonteCarloIntegration(custom_function, a, b, N)

        monte_carlo.visualize_function()

        integration = monte_carlo.calculate_integral()
        print(f"Интеграл методом Монте-Карло: {integration:.4f}")

        exact_integration = monte_carlo.calculate_exact_integral()
        print(f"Точное значение интеграла: {exact_integration:.4f}")

        error = abs(exact_integration - integration) / exact_integration *
100
        print(f"Относительная ошибка: {error:.4f}%")

        print("\n========================================\n")
```

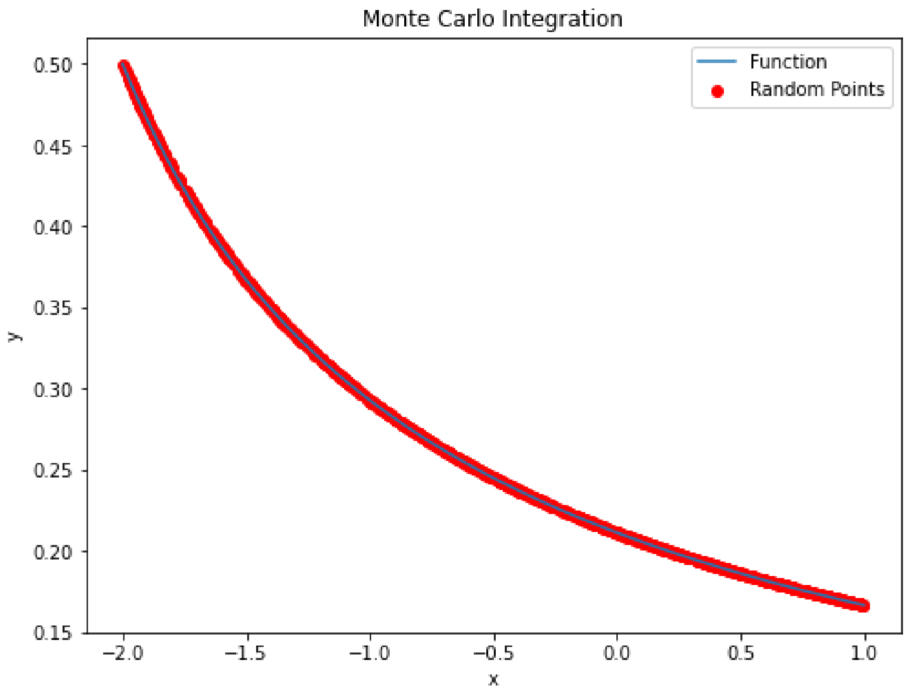
```
Введите нижний предел: -2
Введите верхний предел: 1
Количество рандомных чисел: 10
```

Вывод решений:




```
Интеграл методом Монте-Карло: 0.6844
Точное значение интеграла: 0.8109
Относительная ошибка: 15.5992%
```

```
Количество рандомных чисел: 1000
```




```
Интеграл методом Монте-Карло: 0.8136
Точное значение интеграла: 0.8109
Относительная ошибка: 0.3240%
```

Двойной интеграл с двумя переменными

Если x_i принадлежит [a,b], а y [c,d], где I = 0,1,…, n, то также можно найти интеграл по формуле

$$\iint_{a,c}^{b,d} f(x,y)dxdy = \frac{(b-a)*(d-c)}{N}\sum_{i=0}^{N} f(x_i, y_i)$$

Для примера возьмем следующий интеграл:

$$\iint\limits_{1,2}^{5,10} \sqrt{5x^2-3x+5} + 13\ln(y) + 7\cos(xy)\,dxdy$$

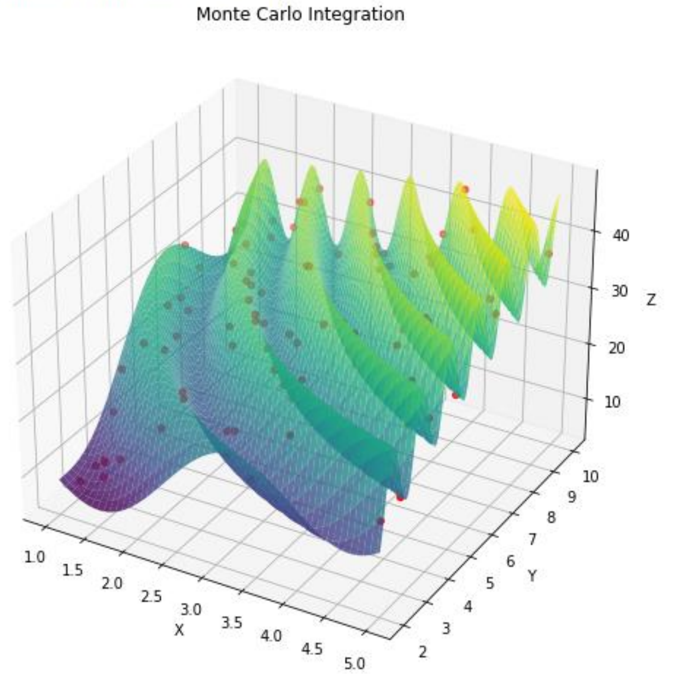
Пример кода:

```
class MonteCarloIntegration:
    def __init__(self, func, a, b, c, d, N):
        self.func = func
        self.a = a
        self.b = b
        self.c = c
        self.d = d
        self.N = N
        self.array_x = np.random.uniform(a, b, N)
        self.array_y = np.random.uniform(c, d, N)

    def calculate_integral(self):
        summa = np.sum(self.func(self.array_x, self.array_y))
        integration = (self.b - self.a) * (self.d - self.c) / self.N * summa
        return integration

    def calculate_exact_integral(self):
        result, _ = nquad(self.func, [[self.a, self.b], [self.c, self.d]])
        return result

    def visualize_function(self):
        x_vals = np.linspace(self.a, self.b, 100)
        y_vals = np.linspace(self.c, self.d, 100)
        X, Y = np.meshgrid(x_vals, y_vals)
        Z = self.func(X, Y)
        fig = plt.figure(figsize=(8, 10))
        ax = fig.add_subplot(111, projection='3d')
        ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.8)
        ax.scatter(self.array_x, self.array_y, self.func(self.array_x,
self.array_y), color='red', label='Random Points')

        ax.set_title('Monte Carlo Integration')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.show()

if __name__ == "__main__":

    def custom_function(x, y):
        return np.sqrt(5*x**2 - 3*x + 5) + 13*np.log(y) + 7*np.cos(x*y)

    a = float(input("Введите нижний предел для x: "))
    b = float(input("Введите верхний предел для x: "))
    c = float(input("Введите нижний предел для y: "))
    d = float(input("Введите верхний предел для y: "))

    N_s = [10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000]

    for N in N_s:

        print("Количество рандомных чисел:", N)

        monte_carlo = MonteCarloIntegration(custom_function, a, b, c, d, N)

        monte_carlo.visualize_function()
```

```
    integration = monte_carlo.calculate_integral()
    print(f"Интеграл методом Монте-Карло: {integration:.4f}")

    exact_integration = monte_carlo.calculate_exact_integral()
    print(f"Точное значение интеграла: {exact_integration:.4f}")

    error = abs(exact_integration - integration) / exact_integration * 100
    print(f"Относительная ошибка: {error:.4f}%")

    print("\n==========================================\n")
```

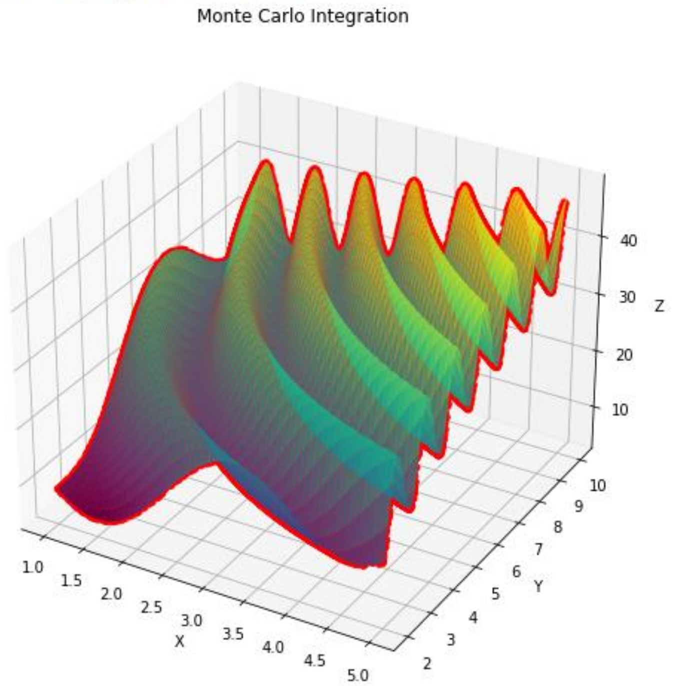
Вывод решений:

```
Количество рандомных чисел: 100
```




```
Интеграл методом Монте-Карло: 913.5059
Точное значение интеграла: 915.5070
Относительная ошибка: 0.2186%

========================================
```

```
Количество рандомных чисел: 100000
```




```
Интеграл методом Монте-Карло: 915.2692
Точное значение интеграла: 915.5070
Относительная ошибка: 0.0260%
```

Кратные интегралы

Расчёт следующих тройных, четверных интегралов, происходит по одной и той же формуле, но с добавлением разницы пар границ в начале дроби и добавлением нового генерирующего числа для расчёта значений функций

Условно, для четверного интеграла, можем применить формулу:

$$\frac{(b-a)*(d-c)*(f-e)*(j-g)}{N}\sum_{i=0}^{N} f(x_i, y_i, z_i, d_i)$$

Решение обыкновенных ДУ

Линейный случай

Пусть нам дано выражение вида:

$$\frac{\partial y}{\partial x} = f(x),\ y(x_0) = y_0$$

Тогда мы можем записать решение, как:

$$y(x) = y(x_0) + \int_{x_0}^{x} \partial\xi f(\xi) = y(x_0) + \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} \partial\xi f(\xi)$$

Где мы определили x_N = x и разделили интеграл на более мелкие дискретные фрагменты, с помощью метода Монте-Карло мы можем это записать как:

$$y(x) = y(x0) + \sum_{i=1}^{N} [\frac{x_i - x_{i-1}}{K} \sum_{k=1}^{K} f(x_k)\ ]$$

Немного упростив, до итераций:

$$y(x_i) = y(x_{i-1}) + \frac{x_i - x_{i-1}}{K} \sum_{k=1}^{K} f(x_k)$$

Решим такой пример:

$$f(x) = x^3 + 2x^2 - 3x$$

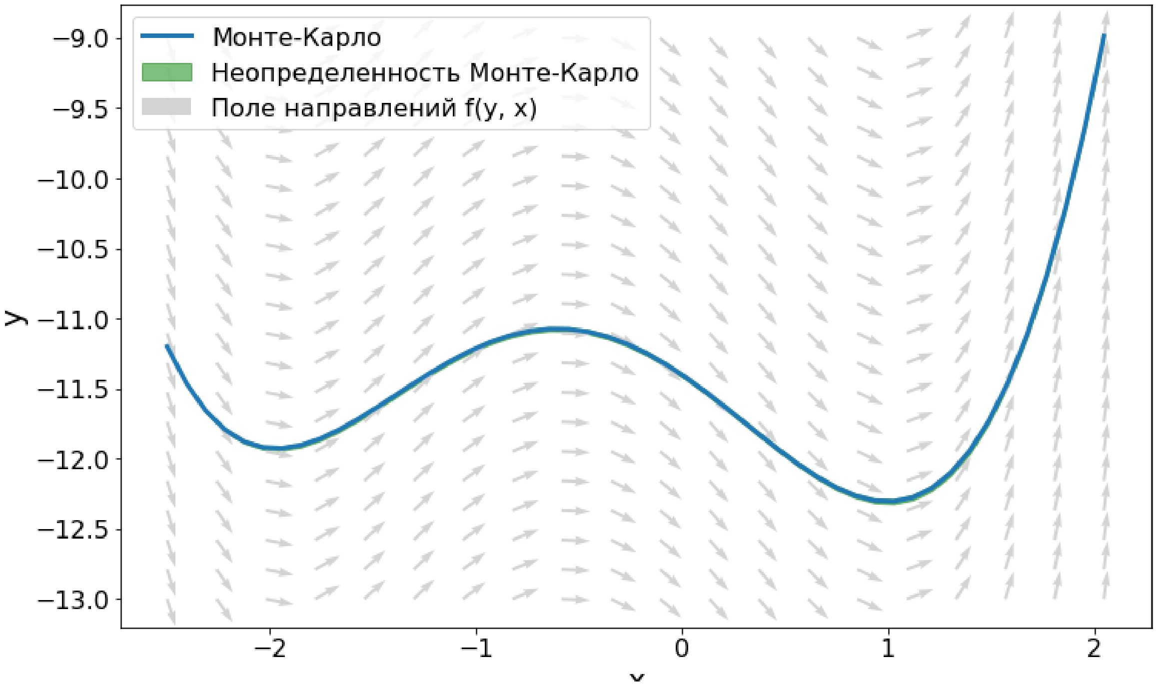
Пример кода:

```
class MonteCarloODESolver:
    def __init__(self, func: Callable):
        self.func = func

    def mc_int(self, domain: Tuple, n_samples: int):
        samples = np.random.uniform(low=domain[0], high=domain[1],
size=(n_samples,))
        volume = abs(domain[1] - domain[0])
        return np.mean(self.func(samples)) * volume

    def solve_ode(self, y0, x, n_samples):
        vals = [y0]
        for lo, hi in zip(x[:-1], x[1:]):
            vals.append(vals[-1] + self.mc_int((lo, hi), n_samples))
        return np.asarray(vals)

    def plot_solution(self, xs, y_mean, y_std):
        width = 12
        height = width / 1.61
        fig, ax = plt.subplots(figsize=(width, height))

        ax.plot(xs, y_mean, linewidth=3, label='Монте-Карло')
        ax.fill_between(xs, y_mean - 3 * y_std, y_mean + 3 * y_std, alpha=.5,
color='g', label='Неопределенность Монте-Карло')

        xx, yy = np.meshgrid(np.linspace(min(xs), max(xs), 20), np.linspace(-
13, -9, 20))
        U = 1
        V = self.func(xx)
        N = np.sqrt(U**2 + V**2)
        U2, V2 = U/N, V/N

        ax.quiver(xx, yy, U2, V2, color='lightgray', label='Поле направлений
f(y, x)')
        ax.set_xlabel(r'x', fontsize=20)
        ax.set_ylabel(r'y', fontsize=20)
        ax.tick_params(axis='both', which='major', labelsize=16)
        ax.tick_params(axis='both', which='minor', labelsize=12)
        ax.legend(loc='upper left', fontsize=16)
        plt.show()

if __name__ == "__main__":

    def func(x):
        return x**3 + 2 * x**2 - 3**x

    xs = np.linspace(-2.5, 2.05, 50)
    y0 = -11.2
    N_s = [10, 50, 100, 500, 1000]

    for N in N_s:

        print("Количество рандомных чисел:", N)

        solver = MonteCarloODESolver(func)
        ys = [solver.solve_ode(y0, xs, N) for _ in range(N)]
        y_mean = np.mean(ys, axis=0)
```

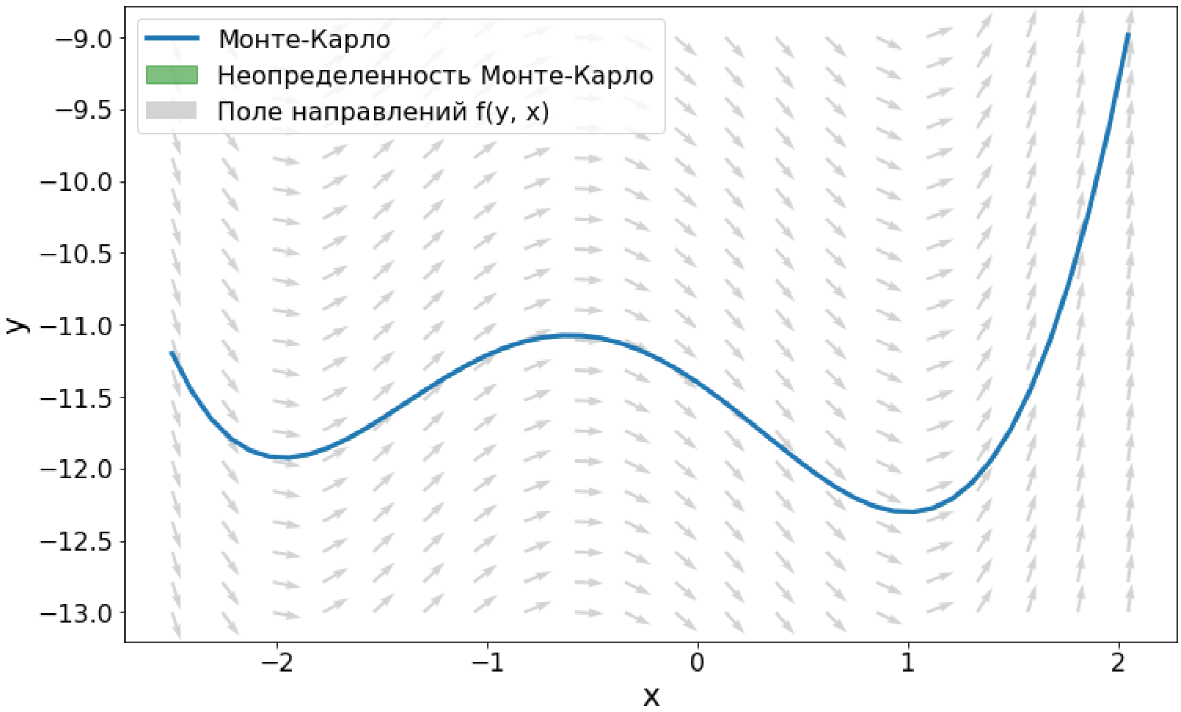
```
y_std = np.std(ys, axis=0)

solver.plot_solution(xs, y_mean, y_std)

print("\n========================================\n")
```

Вывод решений:

Количество рандомных чисел: 50



Количество рандомных чисел: 1000



Как и ожидалось, решение течет вдоль силовых линий направления. Более того, решение с помощью метода Монте-Карло также позволяет нам оценить достоверность решения, что визуализируется зеленой полосой с тремя стандартными отклонениями вокруг решения.

Нелинейный случай

Здесь можно обобщить на нелинейные ОДУ:

$$y(x_i) = y(x_{i-1}) + E_{x \sim U(x_{i-1}, x_i)}[f(y(x), x)]$$

Правая часть теперь явно зависит от y(x), т.е. в случайной точке мы просто производим выборку. Это значение, конечно, недоступно во время вычисления, и нам нужно сделать приближение, чтобы задачу выполнимой. Простое приближение состоит в замене:

$$y(x_i) \to y(x_{i-1}) = y_{i-1}$$

Это приводит к окончательному варианту:

$$y(x_i) = y_{i-1} + E_{x \sim U(x_{i-1}, x_i)}[f(y_{i-1}, x)]$$

Решим следующий пример, нетривиальное градиентное поле:

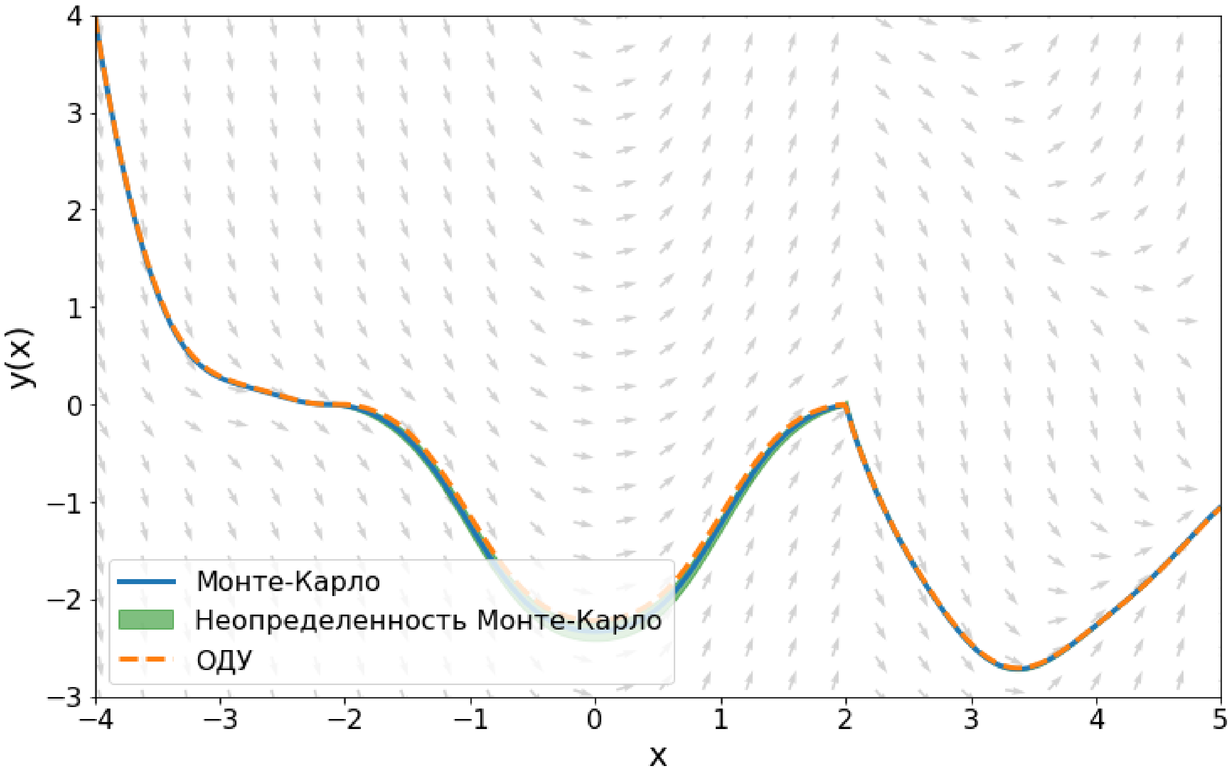
$$f(x, y) = x\sqrt{|y|} + sin^3\left(\frac{pi}{2}x\right) - 5\Theta(x-2)$$

Пример кода:

```
class MonteCarloODESolver:
    def __init__(self, func: Callable):
        self.func = func

    def mc_int(self, func: Callable, domain: Tuple, n_samples: int):
        samples = np.random.uniform(low=domain[0], high=domain[1],
size=(n_samples,))
        volume = abs(domain[1] - domain[0])
        return np.mean(func(samples)) * volume

    def mc_ode_solve(self, y0, t, n_samples=2):
        sols = [y0]
        for lo, hi in zip(t[:-1], t[1:]):
            part_func = lambda v: self.func(x=v, y=sols[-1])
            assert lo < hi
            sols.append(sols[-1] + self.mc_int(part_func, (lo, hi),
n_samples=n_samples))
        return np.asarray(sols)

    def plot_solution(self, base, ys_mc, y_ode, y_mc_mean, y_mc_std):
        width = 12
        height = width / 1.61
        fig, ax = plt.subplots(figsize=(width, height))

        xx, yy = np.meshgrid(base, base)
        U = 1
        V = self.func(yy, xx)
        N = np.sqrt(U**2 + V**2)
        U2, V2 = U/N, V/N

        ax.quiver(xx, yy, U2, V2, color='lightgray')

        y_mc_mean = np.mean(ys_mc, axis=0)
        y_mc_std = np.std(ys_mc, axis=0)

        ax.plot(base2, y_mc_mean, linewidth=3, label='Монте-Карло')
        ax.fill_between(base2, y_mc_mean - 3 * y_mc_std, y_mc_mean + 3 *
y_mc_std, color='g', alpha=0.5, label='Неопределенность Монте-Карло')

        y_ode = odeint(self.func, y0, base2)
        ax.plot(base2, y_ode, '--', linewidth=3, label='ОДУ')

        ax.set_xlabel(r'x', fontsize=20)
        ax.set_ylabel(r'y(x)', fontsize=20)
        ax.tick_params(axis='both', which='major', labelsize=16)
        ax.tick_params(axis='both', which='minor', labelsize=12)

        ax.legend(loc='lower left', fontsize=16)
        ax.set_xlim([-4, 5])
        ax.set_ylim([-3, 4])

        plt.show()

if __name__ == "__main__":
    def func(y, x):
        return x * np.sqrt(np.abs(y)) + np.sin(x * np.pi/2)**3 - 5 * (x > 2)
```

```
base2 = np.linspace(-4, 5, 500)
y0 = 4.
ys_mc = []
for _ in range(20):
    ys_mc.append(MonteCarloODESolver(func).mc_ode_solve(y0, base2))

base = np.linspace(-5, 5, 30)
y_ode = odeint(func, y0, base2)

MonteCarloODESolver(func).plot_solution(base, ys_mc, y_ode, None, None)
```

Вывод результата:



Этот метод работает удивительно хорошо, что можно увидеть по сравнению с модулем scipy.integrate.odeintscipy.integrate.odeint. На рисунке показана абсолютная разница между решениями, которая достаточно мала по сравнению с абсолютным масштабом.

Итог

В рамках курсовой мы изучили базовые принципы метода Монте-Карло, исследовали его на предмет вычисления им интегралов и решению ОДУ.

Помимо самого отчета, есть электронная страница с описанием нашего решения, кода, он будет размещен во вложении курсовой.

Приложение

Курсовая Численные Методы.h

Курсовая Численные Методы.i