МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

МОДЕЛИРОВАНИЕ

ОТЧЁТ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Устюшина Богдана Антоновича

Проверено:

доцент, к. ф.-м. н.	____________	И. Е. Тананко

Саратов 2025

ВВЕДЕНИЕ

Отчёт по предмету моделирование. Всего выполнено 6 заданий: для каждого из них построена математическая модель. Затем на языке программирования Python произведено моделирование с определёнными параметрами, заданными заранее.

Порядковый номер в группе — 15, соответственно все выполненные задачи — пятнадцатые в списке.

1 Моделирование непрерывных систем

1.1 Дифференциальные уравнения первого порядка

Задание

Задача 15. Динамику биомассы M изолированной популяции можно описать уравнением [1]

$$\frac{dM}{dt} = \mu \frac{MS}{K+S} - \varepsilon M,$$

где ε – коэффициент смертности, S – концентрация в среде ресурса питания, μ и K – коэффициенты.

Построить график зависимости функции M от времени t.

Рисунок 1 – Задание 1.1

Решение представлено в приложении 1.1

Результат

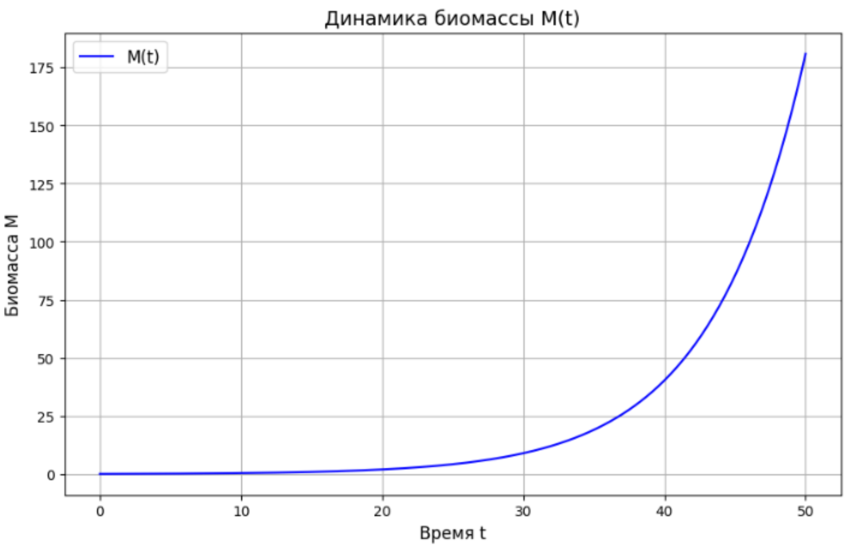
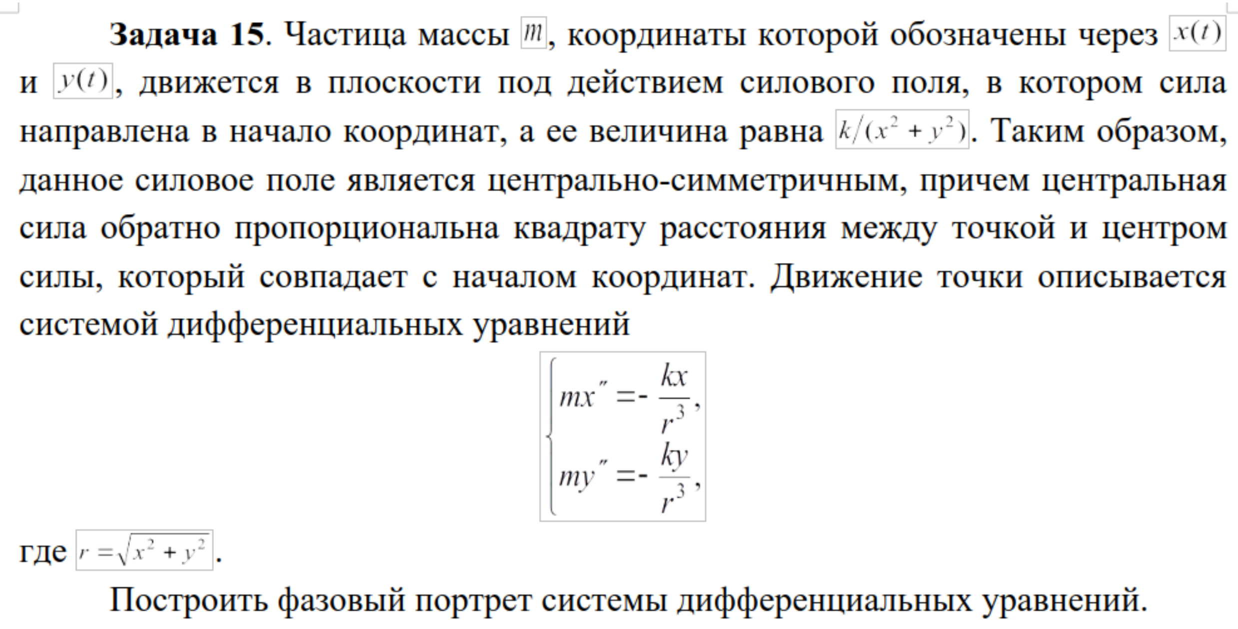


Рисунок 2 – График решения задания 1.1

1.2 Системы дифференциальных уравнений

Задание



Задача 15. Частица массы m, координаты которой обозначены через $x(t)$ и $y(t)$, движется в плоскости под действием силового поля, в котором сила направлена в начало координат, а ее величина равна $k/(x^2+y^2)$. Таким образом, данное силовое поле является центрально-симметричным, причем центральная сила обратно пропорциональна квадрату расстояния между точкой и центром силы, который совпадает с началом координат. Движение точки описывается системой дифференциальных уравнений

$$\begin{cases} mx'' = -\dfrac{kx}{r^3}, \\ my'' = -\dfrac{ky}{r^3}, \end{cases}$$

где $r = \sqrt{x^2+y^2}$.

Построить фазовый портрет системы дифференциальных уравнений.

Рисунок 3 – Задание 1.2

Решение представлено в приложении 1.2

Результат

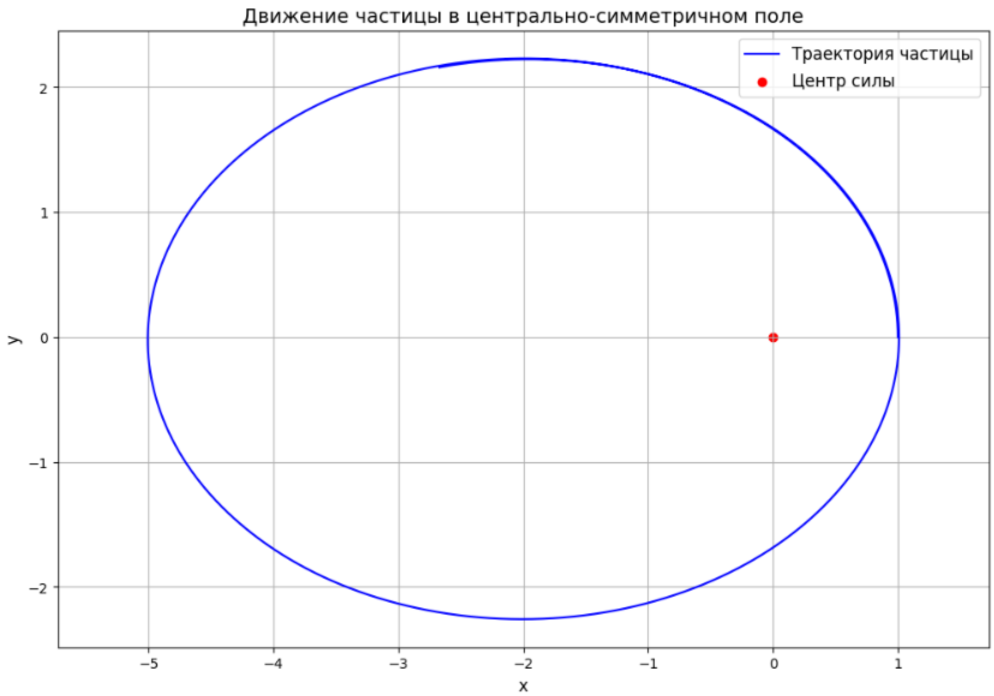


Рисунок 4 – График решения задания 1.2

2 Метод статистических испытаний

2.1 Равномерно распределённая дискретная случайная величина

Задание

Задача 15. Запрос на запасную деталь равен 0, 1, 2 или 3 единицы в день с вероятностями 0,2, 0,3, 0,4 и 0,1 соответственно. Цех технического обеспечения имеет на складе 8 таких деталей и немедленно восстановит запас до этого же уровня, если их останется на складе две или меньше единиц. На основе 1000 испытаний оценить математическое ожидание числа дней до первого пополнения запаса деталей.

Рисунок 5 – Задание 2.1

Решение представлено в Приложении 2.1

Результат

Оценка математического ожидания числа дней до первого пополнения: 4.71

2.2 Равномерно распределённая непрерывная случайная величина

Задание

Задача 15. Два парохода должны подойти к одному и тому же причалу в течение суток. Моменты прихода обоих пароходов есть независимые и равномерно распределенные случайные величины. Используя метод статистических испытаний, оценить вероятность того, что одному из пароходов придется ожидать освобождения причала, если время стоянки первого парохода один час, а второго – два часа.

Рисунок 6 – Задание 2.2

Решение представлено в Приложении 2.2

Результат

Оцененная вероятность ожидания: 0.1195

2.3 Нормально распределённая случайная величина

Задание

Задача 15. В начальный момент времени система состоит из трех новых элементов. Длительность безотказной работы каждого из элементов есть нормально распределенная случайная величина с параметрами μ и σ. При отказе всех элементов система перестает работать. Построить модель возникновения отказов в указанной системе. Провести 1000 испытаний с моделью и оценить математические ожидания: 1) длительности работы элемента, который отказал первым; 2) длительности работы элемента, который отказал вторым; 3) длительности работы элемента, который отказал третьим.

Рисунок 7 – Задание 2.3

Решение представлено в Приложении 2.3

Результат

Математическое ожидание длительности работы элемента, отказавшего первым: 87.62

Математическое ожидание длительности работы элемента, отказавшего вторым: 100.27

Математическое ожидание длительности работы элемента, отказавшего третьим: 112.84

2.4 Экспоненциально распределённая случайная величина

Задание

Задача 15. Система состоит из двух основных элементов и одного резервного. В начальный момент времени начинают работать оба основных элемента. Время жизни основных и резервного элементов – независимые экспоненциально распределенные случайные величины. В момент отказа обоих основных элементов мгновенно включается в работу резервный элемент. С момента отказа резервного элемента вся система перестает работать. Построить имитационную модель этой системы. На основании 1000 испытаний оценить математическое ожидание времени жизни системы и математическое ожидание времени жизни системы до отказа второго основного элемента.

Рисунок 8 – Задание 2.4

Решение представлено в Приложении 2.4

Результат

Математическое ожидание времени жизни системы: 298.75

Математическое ожидание времени жизни системы до отказа второго основного элемента: 150.35

3 Метод статистических испытаний

3.1 Равномерно распределённая дискретная случайная величина

Задание

Задача 15. Дана СМО типа $M \mid M \mid 1$ с двумя классами требований и абсолютным приоритетом. Требования 2-го класса, обслуживание которых было прервано требованиями 1-го класса, мгновенно покидают СМО без дообслуживания. Построить имитационную модель системы. На основании 1000 выборочных значений оценить $\overline{u}$ для каждого класса требований, а также вероятность отказа в обслуживании требований 2-го класса.

Рисунок 9 – Задание 3.1

Решение представлено в Приложении 3.1

Результат

Результаты моделирования (на основе 1000 поступлений):

Класс 1:

Всего поступило = 407, $\overline{u}_1$ (ср. время пребывания) = 0.298

Класс 2:

Всего поступило = 593, $\overline{u}_2$ = 0.792

Вероятность отказа = 0.292

Общее модельное время: 208.852

ПРИЛОЖЕНИЕ А

Решения раздела 1

1.1 Задание 1.1

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
mu = 0.5
K = 1.0
eps = 0.1
S = 1.0
M0 = 0.1
t_span = (0, 50)
t_eval = np.linspace(t_span[0], t_span[1], 500)
def biomass_dynamics(t, M):
    return mu * (M * S) / (K + S) - eps * M
solution = solve_ivp(biomass_dynamics, t_span, [M0], t_eval=t_eval)
plt.figure(figsize=(10, 6))
plt.plot(solution.t, solution.y[0], label="M(t)", color="blue")
plt.title("Динамика биомассы M(t)", fontsize=14)
plt.xlabel("Время t", fontsize=12)
plt.ylabel("Биомасса M", fontsize=12)
plt.grid(True)
plt.legend(fontsize=12)
plt.show()
```

1.2 Задание 1.2

```
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import numpy as np
m = 1.0
k = 1.0
x0, y0 = 1.0, 0.0
vx0, vy0 = 0.0, 1.0
t_span = (0, 50)
```

```
t_eval = np.linspace(t_span[0], t_span[1], 10000)
def central_force(t, state):
    x, y, vx, vy = state
    r = np.sqrt(x**2 + y**2)
    ax = -k * x / r**3
    ay = -k * y / r**3
    return [vx, vy, ax, ay]
initial_state = [x0, y0, vx0, vy0]
solution = solve_ivp(central_force, t_span, initial_state, t_eval=t_eval)
x, y = solution.y[0], solution.y[1]
plt.figure(figsize=(12, 8))
plt.plot(x, y, label="Траектория частицы", color="blue")
plt.scatter(0, 0, color="red", label="Центр силы")
plt.title("Движение частицы в центрально-симметричном поле", fontsize=14)
plt.xlabel("x", fontsize=12)
plt.ylabel("y", fontsize=12)
plt.grid(True)
plt.axis("equal")
plt.legend(fontsize=12)
plt.show()
```

ПРИЛОЖЕНИЕ Б

Решения раздела 2

2.1 Задание 2.1

```python
import numpy as np
probabilities = [0.2, 0.3, 0.4, 0.1]
requests = [0, 1, 2, 3]
initial_stock = 8
threshold = 2
num_trials = 1000
def simulate_single_trial():
    stock = initial_stock
    days = 0
    while stock > threshold:
        daily_request = np.random.choice(requests, p=probabilities)
        stock -= daily_request
        stock = max(stock, 0)
    days += 1
    return days
results = [simulate_single_trial() for _ in range(num_trials)]
expected_days = np.mean(results)
print(f"Оценка математического ожидания числа дней до первого пополнения: {expected_days:.2f}")
```

2.2 Задание 2.2

```python
import numpy as np
time_range = 24
ship1_stay = 1
ship2_stay = 2
num_trials = 100000
def intervals_overlap(start1, duration1, start2, duration2):
    end1 = start1 + duration1
    end2 = start2 + duration2
    return not (end1 <= start2 or end2 <= start1)
conflict_count = 0
```

```
for _ in range(num_trials):
    arrival1 = np.random.uniform(0, time_range)
    arrival2 = np.random.uniform(0, time_range)
    if intervals_overlap(arrival1, ship1_stay, arrival2, ship2_stay):
        conflict_count += 1
probability = conflict_count / num_trials
print(f"Оцененная вероятность ожидания: {probability:.4f}")
```

2.3 Задание 2.3

```
import numpy as np
mean_lifetime = 100
std_lifetime = 15
num_trials = 1000
def simulate_failure_times():
    lifetimes = np.random.normal(mean_lifetime, std_lifetime, 3)
    lifetimes.sort()
    return lifetimes
first_failures = []
second_failures = []
third_failures = []
for _ in range(num_trials):
    lifetimes = simulate_failure_times()
    first_failures.append(lifetimes[0])
    second_failures.append(lifetimes[1])
    third_failures.append(lifetimes[2])
mean_first_failure = np.mean(first_failures)
mean_second_failure = np.mean(second_failures)
mean_third_failure = np.mean(third_failures)
print(f"Математическое ожидание длительности работы элемента, отказавшего первым: {mean_first_failure:.2f}")
print(f"Математическое ожидание длительности работы элемента, отказавшего вторым: {mean_second_failure:.2f}")
print(f"Математическое ожидание длительности работы элемента, отказавшего третьим: {mean_third_failure:.2f}")
```

2.4 Задание 2.4

```
import numpy as np
mean_lifetime_main = 100
mean_lifetime_reserve = 150
num_trials = 10000
def simulate_system_lifetime():
    main1 = np.random.exponential(mean_lifetime_main)
    main2 = np.random.exponential(mean_lifetime_main)
    reserve = np.random.exponential(mean_lifetime_reserve)
second_main_failure = max(main1, main2)
system_lifetime = second_main_failure + reserve
return system_lifetime, second_main_failure
system_lifetimes = []
second_main_failures = []
for _ in range(num_trials):
    system_lifetime, second_main_failure = simulate_system_lifetime()
    system_lifetimes.append(system_lifetime)
    second_main_failures.append(second_main_failure)
mean_system_lifetime = np.mean(system_lifetimes)
mean_second_main_failure = np.mean(second_main_failures)
print(f"Математическое ожидание времени жизни системы: {mean_system_lifetime:.2f}")
print(f"Математическое ожидание времени жизни системы до отказа второго основного элемента: {mean_second_main_failure:.2f}")
```

ПРИЛОЖЕНИЕ В

Решения раздела 3

3.1 Задание 3.1

```python
import random
# Параметры модели
lambda1 = 2
lambda2 = 3
mu = 5.0      # интенсивность обслуживаний
N = 1000
lambda_total = lambda1 + lambda2  # суммарная интенсивность поступлений
t = 0.0
next_arrival = t + random.expovariate(lambda_total)
# current_job - кортеж (job_class, arrival_time) для заявки, находящейся в обслуживании.
current_job = None
departure_time = float('inf')
# Очереди ожидания: для каждого класса храним кортеж (job_class, arrival_time)
queue1 = []  # для требований 1-го класса
queue2 = []  # для требований 2-го класса
# Счётчики поступлений по классам
total_arrivals = 0
total_arrivals_class1 = 0
total_arrivals_class2 = 0
# Для расчёта математического ожидания времени пребывания (u)
sum_sojourn_class1 = 0.0  # суммарное время пребывания для требований 1-го класса
sum_sojourn_class2 = 0.0  # суммарное время пребывания для требований 2-го класса
count_class1 = 0
count_class2 = 0
lost2_count = 0
```

```python
    # Моделирование продолжается, пока не сгенерируем N поступлений и система не опустеет.
    while total_arrivals < N or current_job is not None or queue1 or queue2:
        # Определяем следующее событие: поступление или завершение обслуживания.
        # Если ещё поступления не исчерпаны, выбираем событие с минимальным временем.
        if total_arrivals < N:
            # Сравниваем время следующего поступления и время завершения обслуживания.
            if next_arrival <= departure_time:
                event_type = 'arrival'
                event_time = next_arrival
            else:
                event_type = 'departure'
                event_time = departure_time
        else:
            # Если поступлений больше не генерируем, остаются только события завершения обслуживания.
            event_type = 'departure'
            event_time = departure_time

        t = event_time
        # Обработка поступления: генерируем требование и определяем его класс
        if event_type == 'arrival':
            total_arrivals += 1
            if random.random() < lambda1 / lambda_total:
                job_class = 'class1'
                total_arrivals_class1 += 1
            else:
                job_class = 'class2'
                total_arrivals_class2 += 1

```

```
        arrival_record = (job_class, t)
        # Обработка поступления в зависимости от состояния сервера:
        if current_job is None:
            # Если сервер свободен - начинаем обслуживание сразу.
            current_job = arrival_record
            departure_time = t + random.expovariate(mu)
        else:
            if job_class == 'class1':
                # Требование 1-го класса имеет абсолютный приоритет.
                if current_job[0] == 'class2':
                    # Прерываем обслуживание требования 2-го класса.
                    # Засчитываем время пребывания прерванного требования 2-го класса.
                    sojourn = t - current_job[1]
                    sum_sojourn_class2 += sojourn
                    count_class2 += 1
                    lost2_count += 1
                    # Начинаем обслуживание требования 1-го класса, поступившего в данный момент.
                    current_job = arrival_record
                    departure_time = t + random.expovariate(mu)
                else:
                    # Если сервер занят требованием 1-го класса, поступившее требование идёт в очередь.
                    queue1.append(arrival_record)
            else:
                # Требование 2-го класса: если сервер занят, добавляем в очередь.
```

```
                    queue2.append(arrival_record)

            # Планируем следующее поступление, если общее число поступлений меньше N
            if total_arrivals < N:
                next_arrival = t + random.expovariate(lambda_total)
            else:
                next_arrival = float('inf')
        else:
            # Обработка завершения обслуживания (departure)
            finished_job = current_job
            sojourn = t - finished_job[1]  # время пребывания в системе = t - время поступления
            if finished_job[0] == 'class1':
                sum_sojourn_class1 += sojourn
                count_class1 += 1
            else:
                sum_sojourn_class2 += sojourn
                count_class2 += 1
            # Выбор следующего требования для обслуживания:
            # Абсолютный приоритет имеет очередь требований 1-го класса.
            if queue1:
                current_job = queue1.pop(0)
                departure_time = t + random.expovariate(mu)
            elif queue2:
                current_job = queue2.pop(0)
                departure_time = t + random.expovariate(mu)
            else:
                current_job = None
                departure_time = float('inf')
    u1 = sum_sojourn_class1 / count_class1 if count_class1 > 0 else 0.0
```

```
    u2 = sum_sojourn_class2 / count_class2 if count_class2 > 0 else 0.0
    refusal_prob_class2 = lost2_count / total_arrivals_class2 if total_arrivals_class2 > 0 else 0.0
    # Вывод результатов моделирования
    print("Результаты моделирования (на основе {} поступлений):".format(N))
    print("Класс 1: Всего поступило = {}, u1 (ср. время пребывания) = {:.3f}".format(total_arrivals_class1, u1))
    print("Класс 2: Всего поступило = {}, u2 (ср. время пребывания) = {:.3f}, Вероятность отказа = {:.3f}"
          .format(total_arrivals_class2, u2, refusal_prob_class2))
    print("Общее модельное время: {:.3f}".format(t))
```