

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению практических работ
по дисциплине «Проектирование и анализ экспериментов» для студентов
направления подготовки 09.04.04 Программная инженерия
Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

ЛАБОРАТОРНАЯ РАБОТА №1

Тема: Первичная обработка и разведочный анализ данных (EDA)

Время выполнения: 4 академических часа

Цель: освоить методы первичной обработки, визуализации и проверки статистических предпосылок для последующего планирования эксперимента.

1. Актуальность для программной инженерии

Перед тем как планировать эксперимент (например, сравнение производительности алгоритмов или настройку параметров сервера), необходимо:

- убедиться в **качестве** исходных данных (нет ли пропусков, выбросов);
- проверить **предпосылки** статистических методов (нормальность, однородность дисперсий);
- выполнить **разведочный анализ**, чтобы выдвинуть первичные гипотезы.

В данной работе используется **реальный датасет с логами веб-сервера** (время ответа в мс).

2. Теоретическая справка (кратко)

Метод / тест	Назначение	Критерий принятия решения
Box-plot	Визуальное обнаружение выбросов	Точки за границами $Q1 - 1.5 \cdot IQR$ и $Q3 + 1.5 \cdot IQR$
Гистограмма + Q-Q plot	Визуальная проверка нормальности	Точки на Q-Q должны лечь на прямую
Shapiro-Wilk	Статистический тест на нормальность (при $n \leq 5000$)	$p\text{-value} > 0.05 \rightarrow$ не отвергаем гипотезу о нормальности
Kolmogorov-Smirnov	Сравнение эмпирического распределения с теоретическим	$p\text{-value} > 0.05 \rightarrow$ распределение не отличается от заданного
Skewness (коэфф.	Оценка	0 – симметрично; >0 –

Метод / тест	Назначение	Критерий принятия решения
асимметрии)	симметричности	правосторонняя асимметрия; < 0 – левосторонняя
Kurtosis (коэфф. эксцесса)	Острота пика	3 – нормальное (для некоторых библиотек вычитают 3)

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – инженер по производительности. Система мониторинга собрала время ответа (response_time_ms) эндпоинта `/api/search` за последние 7 дней. Данные записаны в CSV-файл.

Перед вами стоит задача:

1. Загрузить данные и провести их очистку.
2. Выполнить разведочный анализ (EDA) для выявления выбросов, типа распределения.
3. Проверить, можно ли применять параметрические тесты (t-test, ANOVA) для сравнения времени ответа до и после изменения кода.
4. Сформулировать заключение о качестве данных и необходимости преобразований.

4. Исходные данные (генерация или файл)

Создайте файл `server_logs.csv` (или сгенерируйте прямо в ноутбуке). Ниже дан код для генерации (можно выполнить перед началом работы).

```
python
import numpy as np
import pandas as pd

np.random.seed(42)
n = 1000
# смесь: 95% логнормальное (тяжёлый хвост) + 5% выбросы
normal_part = np.random.lognormal(mean=3.5, sigma=0.6, size=int(0.95*n))
outliers = np.random.uniform(2000, 5000, size=int(0.05*n))
times = np.concatenate([normal_part, outliers])
```

```

np.random.shuffle(times)

df = pd.DataFrame({
    'timestamp': pd.date_range('2025-03-01', periods=n, freq='10T'),
    'response_time_ms': times,
    'endpoint': '/api/search',
    'server': np.random.choice(['srv-a', 'srv-b'], size=n)
})

# внесём немного пропусков
df.loc[10:15, 'response_time_ms'] = np.nan

df.to_csv('server_logs.csv', index=False)
print("Файл server_logs.csv создан")

```

5. Ход выполнения (пошаговое руководство)

Выполняйте код в **Jupyter Notebook** или **Google Colab**, фиксируя выводы в Markdown-ячейках.

Шаг 1. Импорт библиотек и загрузка данных

```

python

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

# Настройка стиля
sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 5)

# Загрузка
df = pd.read_csv('server_logs.csv', parse_dates=['timestamp'])
print("Первые 5 строк:")
display(df.head())
print(f"\nРазмер данных: {df.shape}")
print(f"Типы столбцов:\n{df.dtypes}")

```

Шаг 2. Первичная обработка (пропуски, дубликаты)

```
python
# Пропуски
print("Количество пропусков в каждом столбце:")
print(df.isnull().sum())

# Удалим строки с пропуском response_time_ms (для чистоты анализа)
df_clean = df.dropna(subset=['response_time_ms']).copy()
print(f"\nПосле удаления пропусков: {df_clean.shape[0]} записей")

# Дубликаты (в нашем случае маловероятны, но проверим)
print(f"Полных дубликатов: {df_clean.duplicated().sum()}")
```

Шаг 3. Базовые описательные статистики

```
python
desc = df_clean['response_time_ms'].describe(percentiles=[.01, .05, .25, .5, .75, .95, .99])
print("Описательные статистики для response_time_ms:")
print(desc)

# Дополнительные показатели
skew_val = df_clean['response_time_ms'].skew()
kurt_val = df_clean['response_time_ms'].kurtosis() # эксцесс (относительно нормально
zo = 0)
print(f"\nКоэффициент асимметрии (Skewness): {skew_val:.3f}")
print(f"Коэффициент эксцесса (Kurtosis): {kurt_val:.3f}")
```

Вопрос к студенту: Что говорят skewness и kurtosis о форме распределения?

Шаг 4. Визуальный анализ распределения

```
python
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Гистограмма + плотность
sns.histplot(df_clean['response_time_ms'], bins=40, kde=True, ax=axes[0,0])
axes[0,0].set_title('Гистограмма с KDE')
axes[0,0].set_xlabel('Время ответа (мс)')

# Box-plot
sns.boxplot(x=df_clean['response_time_ms'], ax=axes[0,1])
axes[0,1].set_title('Box-plot (выбросы)')

# Q-Q plot
```

```
stats.probplot(df_clean['response_time_ms'], dist="norm", plot=axes[1,0])
axes[1,0].set_title('Q-Q plot относительно нормального распределения')
```

```
# Логарифмическое преобразование (для сравнения)
log_data = np.log(df_clean['response_time_ms'])
stats.probplot(log_data, dist="norm", plot=axes[1,1])
axes[1,1].set_title('Q-Q plot после log-трансформации')
```

```
plt.tight_layout()
plt.show()
```

Вопрос: Почему точки на Q-Q plot исходных данных отклоняются от прямой? Что дало log-преобразование?

Шаг 5. Обнаружение и анализ выбросов

```
python
# Метод межквартильного размаха (IQR)
Q1 = df_clean['response_time_ms'].quantile(0.25)
Q3 = df_clean['response_time_ms'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers_iqr = df_clean[(df_clean['response_time_ms'] < lower_bound) |
                        (df_clean['response_time_ms'] > upper_bound)]
print(f"Количество выбросов по IQR: {len(outliers_iqr)} ({len(outliers_iqr)/len(df_clean)*100:.1f}%)")
print(f"Границы: нижняя = {lower_bound:.1f}, верхняя = {upper_bound:.1f}")
print(f"Максимальное значение: {df_clean['response_time_ms'].max():.1f}")
```

Шаг 6. Проверка статистических предпосылок

```
python
# Тест Шапиро-Уилка (работает при n <= 5000, у нас 1000)
shapiro_stat, shapiro_p = stats.shapiro(df_clean['response_time_ms'])
print(f"Shapiro-Wilk test: statistic={shapiro_stat:.4f}, p-value={shapiro_p:.4e}")

# Тест Колмогорова-Смирнова (сравнение с нормальным распределением)
# Нормируем данные
z_scores = (df_clean['response_time_ms'] - df_clean['response_time_ms'].mean()) / df_clean['response_time_ms'].std()
ks_stat, ks_p = stats.kstest(z_scores, 'norm')
```

```

print(f"Kolmogorov-Smirnov test (vs normal): statistic={ks_stat:.4f}, p-value={ks_p:.4e}")

# Вывод
alpha = 0.05
if shapiro_p > alpha:
    print("\nРезультат: распределение не отличается от нормального (p > 0.05)")
else:
    print("\nРезультат: распределение значимо отличается от нормального (p < 0.05)")

```

Шаг 7. Анализ по категориальным признакам (если есть)

```

python
# Сравним время ответа для разных серверов
print("Статистика по серверам:")
display(df_clean.groupby('server')['response_time_ms'].describe())

# Box-plot по серверам
plt.figure(figsize=(8,4))
sns.boxplot(data=df_clean, x='server', y='response_time_ms')
plt.title('Распределение времени ответа по серверам')
plt.show()

# Проверка равенства дисперсий (тест Левена) – предпосылка ANOVA
levene_stat, levene_p = stats.levene(
    df_clean[df_clean['server']=='srv-a']['response_time_ms'],
    df_clean[df_clean['server']=='srv-b']['response_time_ms']
)
print(f"Тест Левена: p-value = {levene_p:.4f}")

```

Шаг 8. Итоговый вывод о качестве данных и трансформациях

```

python
# Рекомендация по преобразованию
if skew_val > 1 or skew_val < -1:
    print("Сильная асимметрия. Рекомендуется логарифмическое преобразование или Box-Cox.")

# Пример Box-Cox (только для положительных данных)
from scipy.stats import boxcox
transformed, lambda_ = boxcox(df_clean['response_time_ms'])
print(f"Оптимальная лямбда для Box-Cox: {lambda_:.4f}")

```

6. Индивидуальные задания (варианты)

Каждый студент получает **один из вариантов** (меняется датасет или условие).

Задание: выполнить EDA по тому же плану и написать заключение.

Вариант	Характеристика данных
1	Время ответа базы данных (нормальное распределение, выбросов нет)
2	Время ответа API (логнормальное, 10% выбросов сверху)
3	Количество ошибок HTTP (распределение Пуассона)
4	Потребление памяти процессом (бимодальное распределение)
5	Время запуска контейнера (тяжёлый хвост, выбросы в обе стороны)

Для варианта 2–5 можно либо сгенерировать данные через `np.random` согласно нужному распределению, либо использовать встроенные датасеты.

7. Отчет по лабораторной работе

Структура отчета:

1. **Цель работы**
2. **Описание исходных данных** (размер, источники, наличие пропусков)
3. **Результаты выполнения шагов** (скриншоты графиков, таблицы, выводы)
4. **Ответы на контрольные вопросы**
5. **Заключение** (можно ли применять параметрические тесты? какое преобразование выбрать?)
6. **Ссылка на код** (GitHub или встроенный в отчёт)

8. Контрольные вопросы (для защиты)

1. Что такое EDA и почему он важен перед планированием эксперимента?
2. Как интерпретировать Q-Q plot? Приведите примеры отклонения от нормальности.
3. В чём разница между тестом Шапиро-Уилка и Колмогорова-Смирнова?
4. Какой метод обнаружения выбросов используется в box-plot и на чём он основан?
5. Что делать, если распределение не является нормальным, а параметрический тест требуется?

6. Как влияет асимметрия на доверительные интервалы среднего?

ЛАБОРАТОРНАЯ РАБОТА №2

Тема: Сравнение двух выборок (параметрические и непараметрические тесты)

Время выполнения: 4 академических часа

Цель: освоить методы проверки статистических гипотез о различии двух выборок, научиться выбирать между параметрическими (t-test) и непараметрическими (Mann-Whitney, Wilcoxon) критериями в зависимости от предпосылок.

1. Актуальность для программной инженерии

В процессе разработки ПО часто возникает необходимость сравнить две альтернативы:

- время сборки проекта на **CI-runner A** vs **CI-runner B**;
- время ответа API **до** и **после** оптимизации;
- потребление памяти **алгоритмом X** vs **алгоритмом Y**;
- количество ошибок при **A/B-тестировании** двух версий интерфейса.

Правильный выбор статистического критерия позволяет сделать обоснованный вывод: улучшение действительно значимо или различия вызваны случайностью.

2. Теоретическая справка (выбор критерия)

Характеристика	Параметрический t-test	Непараметрический Mann-Whitney U
Тип данных	Количественные, непрерывные	Порядковые, количественные (любое распределение)
Предпосылки	Нормальность, гомогенность дисперсий, независимость	Независимость выборок (ранги)
Мощность	Выше при выполнении предпосылок	Почти такая же при нормальности, выше при выбросах
Выборки	Независимые или зависимые (парный t-test)	Независимые (M-W) или зависимые (Wilcoxon signed-rank)

Характеристика	Параметрический t-test	Непараметрический Mann-Whitney U
Чувствительность к выбросам	Высокая	Низкая

Алгоритм выбора:

1. Проверить нормальность (Shapiro-Wilk / Q-Q plot).
2. Проверить равенство дисперсий (Levene / Bartlett).
3. Если **обе предпосылки выполнены** → **параметрический t-test**.
4. Если нарушены → **Mann-Whitney U** (независимые выборки) или **Wilcoxon** (зависимые).

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – инженер по производительности. Компания переходит с **CI-runner на базе CPU Intel Xeon** на новый **CI-runner на базе AMD EPYC**. Вам предоставили замеры времени сборки (в секундах) одного и того же проекта на 50 запусках на старом и 50 запусках на новом раннере.

Необходимо:

1. Проверить предпосылки (нормальность, равенство дисперсий).
2. Выбрать подходящий критерий и проверить гипотезу о различии среднего времени сборки.
3. Сделать вывод: стоит ли мигрировать полностью на новый раннер.

4. Исходные данные (генерация)

Создайте датасет с реалистичными показателями (лёгкая правосторонняя асимметрия + несколько выбросов). Выполните этот код перед началом работы.

```
python
import numpy as np
import pandas as pd

np.random.seed(42)
n = 50

# Старый раннер (Intel) – больше вариативность, редкие выбросы
old_times = np.random.gamma(shape=2.5, scale=6, size=n) + 15 # среднее ~30
```

```

old_times = np.where(np.random.random(n) < 0.05, old_times * 2.5, old_times) # выбра
сы

# Новый раннер (AMD) – в среднем быстрее, стабильнее
new_times = np.random.gamma(shape=3, scale=5, size=n) + 10 # среднее ~25
new_times = np.where(np.random.random(n) < 0.02, new_times * 1.8, new_times)

df = pd.DataFrame({
    'build_time_sec': np.concatenate([old_times, new_times]),
    'runner': ['old_intel']*n + ['new_amd']*n
})
print("Первые 5 строк:")
print(df.head())
print(f"\nБазовые статистики:\n{df.groupby('runner')['build_time_sec'].describe()}")

```

5. Ход выполнения (пошаговое руководство)

Работа выполняется в **Jupyter Notebook** или **Google Colab**.

Шаг 1. Импорт библиотек и загрузка данных

```

python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 5)

# Если данные сгенерированы выше – они уже в df
# Если загружаете из файла:
# df = pd.read_csv('build_times.csv')

# Разделим выборки для удобства
old = df[df['runner'] == 'old_intel']['build_time_sec'].values
new = df[df['runner'] == 'new_amd']['build_time_sec'].values

print(f"Размер старой выборки: {len(old)}")

```

```
print(f"Размер новой выборки: {len(new)}")
```

Шаг 2. Визуальный анализ и проверка предпосылок

```
python
```

```
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
```

```
# Гистограммы
```

```
axes[0,0].hist(old, bins=12, alpha=0.6, label='old_intel', color='red')
```

```
axes[0,0].hist(new, bins=12, alpha=0.6, label='new_amd', color='blue')
```

```
axes[0,0].set_title('Гистограммы времени сборки')
```

```
axes[0,0].legend()
```

```
# Box-plot
```

```
data_to_box = [old, new]
```

```
axes[0,1].boxplot(data_to_box, labels=['old_intel', 'new_amd'])
```

```
axes[0,1].set_title('Box-plot (выбросы, медиана)')
```

```
axes[0,1].set_ylabel('Время (сек)')
```

```
# Q-Q plot для старого раннера
```

```
stats.probplot(old, dist="norm", plot=axes[1,0])
```

```
axes[1,0].set_title('Q-Q plot: old_intel')
```

```
# Q-Q plot для нового раннера
```

```
stats.probplot(new, dist="norm", plot=axes[1,1])
```

```
axes[1,1].set_title('Q-Q plot: new_amd')
```

```
plt.tight_layout()
```

```
plt.show()
```

Шаг 3. Статистические тесты предпосылок

```
python
```

```
alpha = 0.05
```

```
# 1. Тест Шапиро-Уилка на нормальность
```

```
shapiro_old = stats.shapiro(old)
```

```
shapiro_new = stats.shapiro(new)
```

```
print("=== Проверка нормальности (Shapiro-Wilk) ===")
```

```
print(f"old_intel: statistic={shapiro_old.statistic:.4f}, p-value={shapiro_old.pvalue:.4f}")
```

```
print(f"new_and:  statistic={shapiro_new.statistic:.4f}, p-value={shapiro_new.pvalue:.4f}")
```

```
normal_old = shapiro_old.pvalue > alpha
```

```
normal_new = shapiro_new.pvalue > alpha
```

```
print(f"Нормальность: old {'✅' if normal_old else '❌'}, new {'✅' if normal_new else '❌'}")
```

```
# 2. Тест Левена на равенство дисперсий
```

```
levene_stat, levene_p = stats.levene(old, new)
```

```
print(f"\n=== Проверка равенства дисперсий (Levene) ===")
```

```
print(f"statistic={levene_stat:.4f}, p-value={levene_p:.4f}")
```

```
equal_var = levene_p > alpha
```

```
print(f"Дисперсии равны: {'✅' if equal_var else '❌'}")
```

Шаг 4. Выбор и проведение критерия

```
python
```

```
print("\n=== Сравнение двух независимых выборок ===")
```

```
if normal_old and normal_new and equal_var:
```

```
    # Классический t-test Стьюдента
```

```
    t_stat, t_p = stats.ttest_ind(old, new, equal_var=True)
```

```
    method = "Параметрический t-test (Стьюдента)"
```

```
elif normal_old and normal_new and not equal_var:
```

```
    # t-test Уэлча (не предполагает равенства дисперсий)
```

```
    t_stat, t_p = stats.ttest_ind(old, new, equal_var=False)
```

```
    method = "Параметрический t-test Уэлча (разные дисперсии)"
```

```
else:
```

```
    # Непараметрический критерий Манна-Уитни
```

```
    t_stat, t_p = stats.mannwhitneyu(old, new, alternative='two-sided')
```

```
    method = "Непараметрический U-критерий Манна-Уитни"
```

```
print(f"Используемый метод: {method}")
```

```
print(f"Статистика = {t_stat:.4f}, p-value = {t_p:.4f}")
```

```
if t_p < alpha:
```

```
    print("Вывод: Отвергаем H0. Различия статистически значимы ( $p < 0.05$ ).")
```

```
    if np.mean(new) < np.mean(old):
```

```
        print("Новый раннер показывает значимо меньшее время сборки.")
```

```
    else:
```

```
        print("Новый раннер показывает значимо большее время сборки.")
```

```
else:
```

```
print("Вывод: Нет оснований отвергнуть H0. Различия не значимы (p >= 0.05).")
```

Шаг 5. Оценка эффекта (effect size)

```
python
# Cohen's d (для t-test) или r = Z/sqrt(N) для Mann-Whitney
from scipy.stats import norm

if 'mannwhitneyu' in method.lower():
    # приблизительный effect size для Mann-Whitney
    n_total = len(old) + len(new)
    z_score = norm.ppf(t_p/2) if t_p/2 > 0 else norm.ppf(0.0001) # грубо
    r = abs(z_score) / np.sqrt(n_total)
    effect = r
    effect_desc = "r (Mann-Whitney)"
else:
    # Cohen's d
    pooled_std = np.sqrt(((len(old)-1)*np.var(old, ddof=1) + (len(new)-1)*np.var(new,
ddof=1)) / (len(old)+len(new)-2))
    cohen_d = abs(np.mean(old) - np.mean(new)) / pooled_std
    effect = cohen_d
    effect_desc = "Cohen's d"

print(f"\n=== Размер эффекта ({effect_desc}) ===")
print(f"Значение: {effect:.3f}")
if effect < 0.2:
    print("Интерпретация: очень маленький эффект")
elif effect < 0.5:
    print("Интерпретация: маленький эффект")
elif effect < 0.8:
    print("Интерпретация: средний эффект")
else:
    print("Интерпретация: большой эффект")
```

Шаг 6. Дополнительно: зависимые выборки (парный случай)

Для полноты картины – пример, когда выборки зависимы (замеры до/после на одном и том же объекте).

```
python
# Сгенерируем пример: время отклика API до и после оптимизации
np.random.seed(123)
n_paired = 30
before = np.random.normal(120, 15, n_paired) # 120 мс
```

```

after = before - np.random.normal(10, 8, n_paired)    # улучшение

# Проверка нормальности разностей
diff = before - after
_, p_norm_diff = stats.shapiro(diff)

print("\n=== Пример для зависимых выборок ===")
print(f"Среднее до: {np.mean(before):.1f} мс, после: {np.mean(after):.1f} мс")
print(f"Нормальность разностей: p={p_norm_diff:.4f}")

if p_norm_diff > 0.05:
    t_stat_paired, p_paired = stats.ttest_rel(before, after)
    print(f"Парный t-test: p = {p_paired:.4f}")
else:
    w_stat, p_paired = stats.wilcoxon(before, after)
    print(f"Wilcoxon signed-rank: p = {p_paired:.4f}")

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов** (параметры генерации меняются). Необходимо выполнить полный анализ по шагам 1–5 и написать вывод.

Вар.	Характеристика выборок	Цель эксперимента
1	Группа А: нормальное распределение, равные дисперсии. Группа Б: нормальное, равные дисперсии.	Сравнить время ответа API (2 версии)
2	Группа А: нормальное, разные дисперсии. Группа Б: нормальное, разные дисперсии.	Сравнить время запуска контейнера
3	Группа А: логнормальное (выбросы). Группа Б: логнормальное.	Сравнить потребление памяти двух алгоритмов
4	Группа А: гамма-распределение. Группа Б: гамма-распределение + выбросы.	Сравнить время сборки на разных раннерах
5	Зависимые выборки (до/после). Нормальные разности.	Оценить ускорение после рефакторинга

Вар.	Характеристика выборок	Цель эксперимента
6	Зависимые выборки (до/после). Разности не нормальны.	Оценить изменение числа ошибок в логах

Указание по генерации для варианта 3 (пример):

python

```
A = np.random.lognormal(mean=3, sigma=0.5, size=40)
```

```
B = np.random.lognormal(mean=2.8, sigma=0.7, size=40)
```

7. Структура отчета по лабораторной работе

1. **Цель работы**
2. **Описание данных** (размер, тип распределения, источник)
3. **Результаты проверки предпосылок** (таблицы, графики Q-Q, box-plot)
4. **Выбор и применение критерия** (с обоснованием)
5. **Интерпретация p-value и размера эффекта**
6. **Вывод** (принимаем или отвергаем H_0 , практическая значимость)
7. **Ответы на контрольные вопросы**
8. **Ссылка на код** (GitHub или вставка)

8. Контрольные вопросы (для защиты)

1. В каком случае применяется t-test Стьюдента, а в каком – t-test Уэлча?
2. Какие три основные предпосылки должны выполняться для классического t-test?
3. Почему при наличии выбросов лучше использовать критерий Манна-Уитни?
4. Что такое p-value и как его интерпретировать? Чем оно отличается от уровня значимости α ?
5. Что означает «размер эффекта» (Cohen's d, r) и зачем он нужен, если p-value уже маленькое?
6. В чём разница между независимыми и зависимыми (парными) выборками? Приведите ИТ-примеры.
7. Как проверить нормальность, если выборка больше 5000 элементов?

9. Приложение: полный код в одном блоке (для копирования)

python

```
# Полный код лабораторной работы (шаги 1-5) в виде единого скрипта
```

```
# Вставьте в Jupyter Notebook или Colab
```

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Генерация данных (замените под свой вариант)
np.random.seed(42)
old = np.random.gamma(2.5, 6, 50) + 15
new = np.random.gamma(3, 5, 50) + 10

# Визуализация
fig, ax = plt.subplots(1,2,figsize=(12,4))
ax[0].boxplot([old,new], labels=['old', 'new'])
ax[1].hist(old, alpha=0.5, label='old')
ax[1].hist(new, alpha=0.5, label='new')
ax[1].legend()
plt.show()

# Предпосылки
_, p_old = stats.shapiro(old)
_, p_new = stats.shapiro(new)
_, p_var = stats.levene(old, new)
print(f"Shapiro: old p={p_old:.3f}, new p={p_new:.3f}")
print(f"Levene: p={p_var:.3f}")

# Выбор критерия
if p_old > 0.05 and p_new > 0.05 and p_var > 0.05:
    stat, p = stats.ttest_ind(old, new, equal_var=True)
    method = "t-test Student"
elif p_old > 0.05 and p_new > 0.05:
    stat, p = stats.ttest_ind(old, new, equal_var=False)
    method = "t-test Welch"
else:
    stat, p = stats.mannwhitneyu(old, new, alternative='two-sided')
    method = "Mann-Whitney U"

print(f"{method}: statistic={stat:.3f}, p={p:.4f}")
print("Различия значимы" if p < 0.05 else "Различия не значимы")

```

Данная лабораторная работа даёт практические навыки, которые непосредственно используются при A/B-тестировании, сравнении версий ПО и оптимизации производительности.

ЛАБОРАТОРНАЯ РАБОТА №3

Тема: Однофакторный дисперсионный анализ (ANOVA)

Время выполнения: 4 академических часа

Цель: освоить метод однофакторного дисперсионного анализа для сравнения средних трёх и более групп, научиться проверять предпосылки ANOVA и выполнять post-hoc анализ.

1. Актуальность для программной инженерии

В процессе разработки и эксплуатации ПО часто возникает необходимость сравнить **более двух альтернатив**:

- время выполнения трёх алгоритмов сортировки (quick, merge, heap) на одном наборе данных;
- пропускную способность четырёх СУБД (PostgreSQL, MySQL, MongoDB, Cassandra) при одинаковой нагрузке;
- время ответа API при разных уровнях кэширования (без кэша, Redis, Memcached, in-memory);
- потребление памяти тремя разными сборщиками мусора (G1, Shenandoah, ZGC) в JVM.

t-test позволяет сравнить только две группы. При попарном сравнении трёх групп потребовалось бы 3 теста, что увеличивает риск ошибки I рода (ложноположительный результат). **ANOVA** решает эту проблему, проверяя гипотезу о равенстве средних всех групп за один раз.

2. Теоретическая справка (однофакторный ANOVA)

Фактор – категориальная переменная, которая влияет на отклик (например, тип алгоритма).

Отклик – количественная переменная (время, память, пропускная способность).

Гипотезы:

- **H₀:** $\mu_1 = \mu_2 = \dots = \mu_k$ (средние всех групп равны)
- **H₁:** хотя бы одно среднее отличается от других

Идея метода: ANOVA разбивает общую изменчивость данных на две части:

- **Межгрупповая вариация (SSB)** – за счёт различий между группами (эффект фактора).
- **Внутригрупповая вариация (SSW)** – случайная ошибка (разброс внутри групп).

F-статистика:

$$F = (SSB / (k-1)) / (SSW / (N-k))$$

Если F велико → межгрупповая изменчивость превышает случайную → отвергаем H_0 .

Предпосылки ANOVA:

1. **Независимость наблюдений** (внутри групп и между группами).
2. **Нормальность распределения** отклика в каждой группе (или приблизительная нормальность при больших выборках).
3. **Гомогенность (равенство) дисперсий** во всех группах.

Post-hoc анализ: если H_0 отвергнута, нужно выяснить, какие именно группы различаются. Популярные методы: **Tukey HSD** (честная значимая разница), **Bonferroni**, **Scheffé**.

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – инженер по производительности. Ваша команда разрабатывает высоконагруженный сервис обработки данных. Нужно выбрать оптимальный алгоритм сортировки для массива из 1 млн записей. Тестируются три алгоритма:

- **QuickSort** (быстрая сортировка)
- **MergeSort** (сортировка слиянием)
- **HeapSort** (пирамидальная сортировка)

Каждый алгоритм запускается 30 раз на одних и тех же входных данных (случайный массив). Измеряется время выполнения (в миллисекундах).

Задача:

1. Проверить предпосылки ANOVA (нормальность, гомогенность дисперсий).
2. Провести однофакторный ANOVA и определить, есть ли статистически значимые различия между алгоритмами.
3. Если различия есть – выполнить post-hoc анализ (Tukey HSD) и определить, какой алгоритм лучше.
4. Дать рекомендацию команде.

4. Исходные данные (генерация)

Создайте реалистичный датасет (время выполнения в мс). Выполните этот код перед началом работы.

```
python
import numpy as np
import pandas as pd

np.random.seed(42)
n_per_group = 30 # по 30 замеров на алгоритм

# QuickSort - самый быстрый в среднем, но возможны выбросы (случай опорного элемента)
quick = np.random.normal(loc=125, scale=12, size=n_per_group)
quick = np.where(np.random.random(n_per_group) < 0.05, quick * 1.8, quick)

# MergeSort - стабильный, средний
merge = np.random.normal(loc=148, scale=10, size=n_per_group)

# HeapSort - медленнее, но без выбросов
heap = np.random.normal(loc=162, scale=11, size=n_per_group)

# Собираем DataFrame
df = pd.DataFrame({
    'time_ms': np.concatenate([quick, merge, heap]),
    'algorithm': ['QuickSort']*n_per_group + ['MergeSort']*n_per_group + ['HeapSort']*n_per_group
})

print("Первые 5 строк:")
print(df.head())
print("\nБазовые статистики по группам:")
print(df.groupby('algorithm')['time_ms'].describe())
```

5. Ход выполнения (пошаговое руководство)

Работа выполняется в **Jupyter Notebook** или **Google Colab**.

Шаг 1. Импорт библиотек и подготовка данных

```
python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from statsmodels.formula.api import ols
```

```

from statsmodels.stats.multicomp import pairwise_tukeyhsd
from statsmodels.stats.diagnostic import het_breuschpagan
import warnings
warnings.filterwarnings('ignore')

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 5)

# Разделим данные для удобства
groups = {
    'QuickSort': df[df['algorithm'] == 'QuickSort']['time_ms'].values,
    'MergeSort': df[df['algorithm'] == 'MergeSort']['time_ms'].values,
    'HeapSort': df[df['algorithm'] == 'HeapSort']['time_ms'].values
}

print("Размеры групп:")
for name, data in groups.items():
    print(f" {name}: n = {len(data)}")

```

Шаг 2. Визуальный анализ

```

python
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Box-plot
sns.boxplot(data=df, x='algorithm', y='time_ms', ax=axes[0], palette='Set2')
axes[0].set_title('Распределение времени выполнения по алгоритмам')
axes[0].set_ylabel('Время (мс)')
axes[0].set_xlabel('Алгоритм')

# Violin plot (более информативный)
sns.violinplot(data=df, x='algorithm', y='time_ms', ax=axes[1], palette='Set2')
axes[1].set_title('Violin plot: плотность распределения')
axes[1].set_ylabel('Время (мс)')
axes[1].set_xlabel('Алгоритм')

plt.tight_layout()
plt.show()

# Q-Q plots для каждой группы
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
for i, (name, data) in enumerate(groups.items()):
    stats.probplot(data, dist="norm", plot=axes[i])

```

```

        axes[i].set_title(f'Q-Q plot: {name}')
plt.tight_layout()
plt.show()

```

Шаг 3. Проверка предпосылок ANOVA

python

```
alpha = 0.05
```

1. Нормальность (Shapiro-Wilk для каждой группы)

```

print("=== Проверка нормальности (Shapiro-Wilk) ===")
normal_flag = True
for name, data in groups.items():
    stat, p = stats.shapiro(data)
    is_normal = p > alpha
    print(f"{name}: statistic={stat:.4f}, p-value={p:.4f} -> {'✅нормальное' if is_no
normal else '❌не нормальное'}")
    if not is_normal:
        normal_flag = False

```

2. Равенство дисперсий (Levene / Bartlett)

Levene менее чувствителен к отклонениям от нормальности

```

levene_stat, levene_p = stats.levene(*groups.values())
print(f"\n=== Проверка равенства дисперсий (Levene) ===")
print(f"statistic={levene_stat:.4f}, p-value={levene_p:.4f}")
equal_var = levene_p > alpha
print(f"Дисперсии равны: {'✅' if equal_var else '❌'}")

```

Вывод о применимости ANOVA

```

if normal_flag and equal_var:
    print("\n✅Предпосылки ANOVA выполнены. Можно использовать классический ANOVA.")
else:
    print("\n Предпосылки нарушены. Рекомендуется использовать непараметрический ана
лог (Kruskal-Wallis) или преобразование данных.")

```

Шаг 4. Проведение однофакторного ANOVA

python

```
print("\n=== Однофакторный дисперсионный анализ ===")
```

Способ 1: через scipy

```

f_stat, p_value = stats.f_oneway(groups['QuickSort'], groups['MergeSort'], groups['He
apSort'])

```

```

print(f"scipy.f_oneway: F = {f_stat:.4f}, p-value = {p_value:.6f}")

# Способ 2: через statsmodels (даёт больше деталей)
model = ols('time_ms ~ C(algorithm)', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print("\nТаблица ANOVA (statsmodels):")
print(anova_table)

# Вывод
if p_value < alpha:
    print(f"\nРезультат: Отвергаем H0 (p = {p_value:.6f} < {alpha})")
    print("Между алгоритмами есть статистически значимые различия в среднем времени в\nвыполнения.")
else:
    print(f"\nРезультат: Нет оснований отвергнуть H0 (p = {p_value:.4f} >= {alpha})")
    print("Различия между алгоритмами не значимы.")

```

Шаг 5. Post-hoc анализ (Tukey HSD)

Выполняется только если ANOVA показал значимые различия.

```

python
print("\n=== Post-hoc анализ (Tukey HSD) ===")
tukey = pairwise_tukeyhsd(endog=df['time_ms'], groups=df['algorithm'], alpha=0.05)
print(tukey.summary())

# Визуализация результатов Tukey
fig = plt.figure(figsize=(10, 4))
tukey.plot_simultaneous(ax=plt.gca())
plt.title('Tukey HSD: одновременные доверительные интервалы')
plt.xlabel('Время (мс)')
plt.tight_layout()
plt.show()

```

Шаг 6. Оценка размера эффекта (η^2 – eta-squared)

```

python
# Eta squared = SSB / SST (доля дисперсии, объяснённая фактором)
ss_between = anova_table['sum_sq']['C(algorithm)']
ss_total = ss_between + anova_table['sum_sq']['Residual']
eta_squared = ss_between / ss_total

print("\n=== Размер эффекта ===")
print(f"Eta-squared ( $\eta^2$ ) = {eta_squared:.4f}")

```

```

if eta_squared < 0.01:
    print("Интерпретация: очень маленький эффект")
elif eta_squared < 0.06:
    print("Интерпретация: маленький эффект")
elif eta_squared < 0.14:
    print("Интерпретация: средний эффект")
else:
    print("Интерпретация: большой эффект")

```

Шаг 7. Альтернатива: непараметрический Kruskal-Wallis (при нарушении предпосылок)

Для демонстрации – если бы предпосылки не выполнялись.

```

python
print("\n=== Непараметрический аналог ANOVA (Kruskal-Wallis) ===")
h_stat, kw_p = stats.kruskal(groups['QuickSort'], groups['MergeSort'], groups['HeapSort'])
print(f"Kruskal-Wallis H = {h_stat:.4f}, p-value = {kw_p:.6f}")

if kw_p < alpha:
    print("Различия между группами значимы (по рангам).")
    # Post-hoc для Kruskal-Wallis – попарные Mann-Whitney с коррекцией Бонферрони
    print("\nПопарное сравнение (Mann-Whitney с коррекцией Бонферрони):")
    from itertools import combinations
    names = list(groups.keys())
    for (i1, n1), (i2, n2) in zip(combinations(names, 2), combinations(names, 2)):
        _, p_mw = stats.mannwhitneyu(groups[n1], groups[n2], alternative='two-sided')
        p_corrected = p_mw * 3 # 3 сравнения
        print(f" {n1} vs {n2}: raw p={p_mw:.4f}, corrected p={p_corrected:.4f} -> {'различаются' if p_corrected < alpha else 'не различаются'}")

```

Шаг 8. Итоговый вывод и рекомендация

```

python
print("\n" + "="*60)
print("ИТОГОВЫЙ ВЫВОД")
print("="*60)

mean_times = df.groupby('algorithm')['time_ms'].mean().sort_values()
best_algo = mean_times.index[0]
best_time = mean_times.iloc[0]

```

```

print(f"Среднее время выполнения (мс):")
for algo, t in mean_times.items():
    print(f" {algo}: {t:.1f} мс")

if p_value < alpha:
    print(f"\n✅Статистически значимые различия обнаружены (p = {p_value:.5f}).")
    print(f"Рекомендуемый алгоритм: {best_algo} (среднее {best_time:.1f} мс).")
    if 'tukey' in locals():
        print("Попарные сравнения (Tukey HSD) подтверждают, что он значимо лучше друг
их.")
    else:
        print(f"\n❌Статистически значимых различий не обнаружено (p = {p_value:.4f}).")
        print("Выбор алгоритма может основываться на других критериях (сложность реализац
ии, потребление памяти).")

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов** (меняются данные и ИТ-контекст). Необходимо выполнить полный анализ по шагам 1–8.

Вар.	Фактор (группы)	Отклик	Контекст
1	3 СУБД (PostgreSQL, MySQL, MongoDB)	Время выполнения SELECT запроса (мс)	Выбор БД для аналитики
2	4 уровня кэширования (none, Redis, Memcached, local)	Время ответа API (мс)	Оптимизация веб-сервиса
3	3 сборщика мусора JVM (G1, Shenandoah, ZGC)	Средняя пауза GC (мс)	Настройка Java-приложения
4	4 метода сжатия (gzip, brotli, zstd, lz4)	Время сжатия 1 ГБ данных (сек)	Выбор для бэкапов
5	3 формата сериализации (JSON, Protobuf, Avro)	Время сериализации (мкс)	High-load система
6	4 алгоритма балансировки (round-robin, least-conn, random, ip-hash)	Время отклика при 1000 RPS (мс)	Настройка балансировщика

Указание по генерации данных для варианта (пример для варианта 1):

```
python
pg = np.random.normal(45, 6, 30)      # PostgreSQL
mysql = np.random.normal(52, 8, 30)   # MySQL
mongo = np.random.normal(68, 10, 30)  # MongoDB
```

7. Структура отчета

1. **Цель работы**
2. **Описание фактора и отклика** (контекст ИТ-задачи)
3. **Визуальный анализ** (box-plot, violin plot, Q-Q plots)
4. **Проверка предпосылок** (нормальность, гомогенность дисперсий)
5. **Результаты ANOVA** (F-статистика, p-value, таблица)
6. **Post-hoc анализ** (Tukey HSD, визуализация)
7. **Размер эффекта** (η^2)
8. **Вывод и рекомендация** (какая группа лучше, почему)
9. **Ответы на контрольные вопросы**
10. **Код** (ссылка или вставка)

8. Контрольные вопросы (для защиты)

1. Почему нельзя просто выполнить попарные t-тесты для трёх и более групп?
2. Что такое SSB (sum of squares between) и SSW (sum of squares within) и как они интерпретируются?
3. Какие три предпосылки должны выполняться для классического ANOVA?
4. Что показывает F-статистика? Как связаны F и p-value?
5. Для чего нужен post-hoc анализ после значимого ANOVA? Почему не достаточно самого ANOVA?
6. Чем отличается Tukey HSD от попарных t-тестов с коррекцией Бонферрони?
7. Что такое η^2 (eta-squared) и как интерпретировать его значение?
8. Какой непараметрический аналог ANOVA используется при нарушении нормальности?
9. Может ли ANOVA показать значимые различия, если все группы имеют одинаковые средние, но разные дисперсии? (Ответ: нет, ANOVA проверяет средние, но нарушение гомогенности дисперсий повышает риск ошибки.)
10. Приведите ИТ-пример, где нужно сравнить 4 и более группы.

9. Полный код в одном блоке (для копирования)

```
python
# Полный код лабораторной работы (шаги 1-8)
# Вставьте в Jupyter Notebook или Colab
```

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Генерация данных
np.random.seed(42)
n = 30
quick = np.random.normal(125, 12, n)
merge = np.random.normal(148, 10, n)
heap = np.random.normal(162, 11, n)
df = pd.DataFrame({
    'time_ms': np.concatenate([quick, merge, heap]),
    'algorithm': ['Quick']*n + ['Merge']*n + ['Heap']*n
})

# Визуализация
sns.boxplot(data=df, x='algorithm', y='time_ms')
plt.title('Сравнение алгоритмов сортировки')
plt.show()

# Предпосылки
_, p_quick = stats.shapiro(quick)
_, p_merge = stats.shapiro(merge)
_, p_heap = stats.shapiro(heap)
_, p_var = stats.levene(quick, merge, heap)
print(f"Нормальность: Quick p={p_quick:.3f}, Merge p={p_merge:.3f}, Heap p={p_heap:.3f}")
print(f"Равенство дисперсий (Levene): p={p_var:.3f}")

# ANOVA
f_stat, p_val = stats.f_oneway(quick, merge, heap)
print(f"ANOVA: F={f_stat:.3f}, p={p_val:.5f}")

if p_val < 0.05:
    print("Различия значимы. Post-hoc Tukey:")
    tukey = pairwise_tukeyhsd(df['time_ms'], df['algorithm'], alpha=0.05)
    print(tukey)

```

```
else:  
    print("Различия не значимы.")
```

Данная лабораторная работа даёт практические навыки, которые необходимы для сравнения производительности нескольких реализаций, выбора оптимальных параметров и принятия инженерных решений на основе данных.

ЛАБОРАТОРНАЯ РАБОТА №4

Тема: Двухфакторный дисперсионный анализ с взаимодействиями (Two-way ANOVA)

Время выполнения: 4 академических часа

Цель: освоить метод двухфакторного дисперсионного анализа для оценки влияния двух факторов и их взаимодействия на отклик, научиться интерпретировать графики взаимодействий.

1. Актуальность для программной инженерии

В реальных ИТ-системах на производительность влияют **одновременно несколько факторов**, которые могут **взаимодействовать** друг с другом. Примеры:

- Влияние **типа базы данных** (PostgreSQL, MongoDB) и **уровня кэширования** (нет, Redis) на время ответа API.
- Влияние **алгоритма сжатия** (gzip, brotli) и **размера блока данных** (4KB, 64KB) на пропускную способность.
- Влияние **сборщика мусора JVM** (G1, ZGC) и **размера кучи** (2GB, 8GB) на паузы GC.

Важнейший эффект – взаимодействие (interaction):

Эффект одного фактора зависит от уровня другого фактора.

Например: кэширование может ускорять PostgreSQL сильно, а MongoDB – слабо. Это взаимодействие невозможно обнаружить, анализируя каждый фактор по отдельности.

Two-way ANOVA позволяет:

1. Оценить **главные эффекты** (main effects) каждого фактора.
2. Оценить **эффект взаимодействия** (interaction effect).
3. Ответить на вопрос: «Работает ли фактор А одинаково при всех уровнях фактора В?»

2. Теоретическая справка (двухфакторный ANOVA)

Модель (с повторениями):

$$Y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk}$$

Где:

- Y_{ijk} – отклик для i-го уровня фактора A, j-го уровня фактора B, k-го повторения
- μ – общее среднее
- α_i – эффект i-го уровня фактора A
- β_j – эффект j-го уровня фактора B
- $(\alpha\beta)_{ij}$ – эффект взаимодействия
- ε_{ijk} – случайная ошибка

Гипотезы:

Эффект	H_0	H_1
Фактор A	$\alpha_1 = \alpha_2 = \dots = 0$	хотя бы один $\alpha_i \neq 0$ • $\neq 0$
Фактор B	$\beta_1 = \beta_2 = \dots = 0$	хотя бы один $\beta_j \neq 0$ • $\neq 0$
Взаимодействие A×B	$(\alpha\beta)_{ij} = 0$ для всех i, j	хотя бы одно $(\alpha\beta)_{ij} \neq 0$ • $\neq 0$

Предпосылки (те же, что для однофакторного ANOVA):

1. Независимость наблюдений
2. Нормальность остатков (или распределения в каждой ячейке)
3. Гомогенность дисперсий во всех группах (комбинациях уровней факторов)

График взаимодействия – линии, соединяющие средние значения.

- Если линии **параллельны** → взаимодействия нет.
- Если линии **пересекаются или сильно расходятся** → взаимодействие есть.

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – инженер по производительности. Ваш сервис обрабатывает пользовательские запросы. Вы хотите выбрать оптимальную конфигурацию. Два фактора:

- **Фактор A (тип базы данных):** PostgreSQL, MongoDB
- **Фактор B (уровень кэширования):** No cache, Redis, Memcached

Отклик – **время ответа API** (в миллисекундах), измеренное при 10 повторениях для каждой комбинации (сбалансированный дизайн: $2 \times 3 \times 10 = 60$ наблюдений).

Задача:

1. Визуализировать данные (box-plot, график взаимодействий).
2. Проверить предпосылки двухфакторного ANOVA.
3. Выполнить Two-way ANOVA и интерпретировать главные эффекты и взаимодействие.
4. Если взаимодействие значимо – проанализировать его характер.
5. Дать рекомендации по выбору конфигурации.

4. Исходные данные (генерация)

Создайте датасет с **реалистичным взаимодействием**: для PostgreSQL кэширование даёт большой выигрыш, для MongoDB – небольшой.

```
python
import numpy as np
import pandas as pd

np.random.seed(42)

# Параметры
n_replicates = 10 # повторений для каждой комбинации
databases = ['PostgreSQL', 'MongoDB']
cache_levels = ['No cache', 'Redis', 'Memcached']

data = []

# Базовые средние (время ответа в мс)
# PostgreSQL: без кэша 150, Redis 70, Memcached 80
# MongoDB: без кэша 200, Redis 120, Memcached 130
# Эффект взаимодействия: PostgreSQL выигрывает от кэша больше

mean_values = {
    ('PostgreSQL', 'No cache'): 150,
    ('PostgreSQL', 'Redis'): 70,
    ('PostgreSQL', 'Memcached'): 80,
    ('MongoDB', 'No cache'): 200,
    ('MongoDB', 'Redis'): 120,
    ('MongoDB', 'Memcached'): 130
}
```

```

# Стандартное отклонение (гомогенность дисперсий)
std_dev = 12

for db in databases:
    for cache in cache_levels:
        mu = mean_values[(db, cache)]
        times = np.random.normal(loc=mu, scale=std_dev, size=n_replicates)
        # Добавляем редкие выбросы для реалистичности
        times = np.where(np.random.random(n_replicates) < 0.05, times * 1.5, times)
        for t in times:
            data.append({
                'database': db,
                'cache': cache,
                'response_time_ms': max(10, round(t, 1)) # неотрицательное время
            })

df = pd.DataFrame(data)
print("Первые 10 строк:")
print(df.head(10))
print("\nСреднее время ответа по комбинациям:")
print(df.groupby(['database', 'cache'])['response_time_ms'].agg(['mean', 'std', 'count']))

```

5. Ход выполнения (пошаговое руководство)

Работа выполняется в **Jupyter Notebook** или **Google Colab**.

Шаг 1. Импорт библиотек и подготовка данных

```

python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from statsmodels.graphics.factorplots import interaction_plot
import warnings
warnings.filterwarnings('ignore')

```

```

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 5)

# Проверка сбалансированности
print("Количество наблюдений по комбинациям:")
print(df.groupby(['database', 'cache']).size())

```

Шаг 2. Визуальный анализ

```

python
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Box-plot по комбинациям
sns.boxplot(data=df, x='cache', y='response_time_ms', hue='database', ax=axes[0])
axes[0].set_title('Время ответа: кэш и тип БД')
axes[0].set_ylabel('Время ответа (мс)')

# График взаимодействий (средние значения)
# Подготовим таблицу средних
pivot_mean = df.pivot_table(index='database', columns='cache', values='response_time_ms', aggfunc='mean')
for db in pivot_mean.index:
    axes[1].plot(pivot_mean.columns, pivot_mean.loc[db], marker='o', label=db)
axes[1].set_title('График взаимодействий (средние значения)')
axes[1].set_xlabel('Уровень кэширования')
axes[1].set_ylabel('Среднее время ответа (мс)')
axes[1].legend()
axes[1].grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

print("\nТаблица средних:")
print(pivot_mean)

```

Интерпретация: Если линии не параллельны – есть взаимодействие.

Шаг 3. Проверка предпосылок

```

python
alpha = 0.05

# 1. Нормальность остатков (модель с взаимодействием)

```

```

model_full = ols('response_time_ms ~ C(database) * C(cache)', data=df).fit()
residuals = model_full.resid

# Shapiro-Wilk для остатков
shapiro_stat, shapiro_p = stats.shapiro(residuals)
print("=== Проверка нормальности остатков ===")
print(f"Shapiro-Wilk: statistic={shapiro_stat:.4f}, p-value={shapiro_p:.4f}")
if shapiro_p > alpha:
    print("✅ Остатки распределены нормально (предпосылка выполняется)")
else:
    print("❌ Остатки не нормальны (возможно, при большом объёме ANOVA устойчив)")

# Q-Q plot остатков
fig, ax = plt.subplots(figsize=(6,5))
stats.probplot(residuals, dist="norm", plot=ax)
ax.set_title('Q-Q plot остатков')
plt.show()

# 2. Гомогенность дисперсий (Levene по группам)
# Создаём группирующую переменную
df['group'] = df['database'] + '_' + df['cache']
groups = [df[df['group'] == g]['response_time_ms'].values for g in df['group'].unique()]

levene_stat, levene_p = stats.levene(*groups)
print("\n=== Проверка равенства дисперсий (Levene) ===")
print(f"statistic={levene_stat:.4f}, p-value={levene_p:.4f}")
if levene_p > alpha:
    print("✅ Дисперсии однородны (предпосылка выполняется)")
else:
    print("❌ Дисперсии неоднородны (рекомендуется преобразование или Welch ANOVA)")

# 3. Визуализация остатков vs предсказанные значения
fitted = model_full.fittedvalues
fig, ax = plt.subplots(figsize=(8,5))
ax.scatter(fitted, residuals, alpha=0.6)
ax.axhline(y=0, color='red', linestyle='--')
ax.set_xlabel('Предсказанные значения')
ax.set_ylabel('Остатки')
ax.set_title('Остатки vs Предсказанные значения')
ax.grid(True, linestyle='--', alpha=0.3)
plt.show()

```

Шаг 4. Двухфакторный ANOVA (с взаимодействием)

```
python
print("\n" + "="*60)
print("ДВУХФАКТОРНЫЙ ANOVA (с взаимодействием)")
print("="*60)

# Модель с взаимодействием: response ~ factorA * factorB
model = ols('response_time_ms ~ C(database) * C(cache)', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=2) # тип 2 - для сбалансированного дизайна
print(anova_table)

# Извлечём p-value
p_db = anova_table['PR(>F)']['C(database)']
p_cache = anova_table['PR(>F)']['C(cache)']
p_interaction = anova_table['PR(>F)']['C(database):C(cache)']

print("\n=== Интерпретация ===")
print(f"Фактор 'Тип БД': p = {p_db:.6f} -> {'✅значим' if p_db < alpha else '❌не значим'}")
print(f"Фактор 'Кэширование': p = {p_cache:.6f} -> {'✅значим' if p_cache < alpha else '❌не значим'}")
print(f"Взаимодействие БД × Кэш: p = {p_interaction:.6f} -> {'✅значимо' if p_interaction < alpha else '❌не значимо'}")
```

Шаг 5. Анализ взаимодействия (если оно значимо)

```
python
if p_interaction < alpha:
    print("\n" + "="*60)
    print("АНАЛИЗ ВЗАИМОДЕЙСТВИЯ")
    print("="*60)

    # Простой эффект: влияние кэша отдельно для каждой БД
    print("Простой эффект кэширования для каждой БД:")
    for db in df['database'].unique():
        sub_df = df[df['database'] == db]
        # Однофакторный ANOVA для этой БД
        groups_cache = [sub_df[sub_df['cache'] == c]['response_time_ms'].values for c
in cache_levels]
        f_stat, p_val = stats.f_oneway(*groups_cache)
```

```

        print(f" {db}: F = {f_stat:.2f}, p = {p_val:.6f} -> {'значимо' if p_val < 0.
05 else 'не значимо'}")

# Попарные сравнения внутри уровней (Tukey HSD для каждой БД отдельно)
print("\nПопарные сравнения (Tukey HSD) для PostgreSQL:")
pg_df = df[df['database'] == 'PostgreSQL']
tukey_pg = pairwise_tukeyhsd(pg_df['response_time_ms'], pg_df['cache'], alpha=0.0
5)
print(tukey_pg)

print("\nПопарные сравнения (Tukey HSD) для MongoDB:")
mongo_df = df[df['database'] == 'MongoDB']
tukey_mongo = pairwise_tukeyhsd(mongo_df['response_time_ms'], mongo_df['cache'],
alpha=0.05)
print(tukey_mongo)

# Визуализация взаимодействия с доверительными интервалами
fig, ax = plt.subplots(figsize=(8,6))
# Рассчитаем стандартные ошибки
means = df.groupby(['database', 'cache'])['response_time_ms'].mean().unstack()
errors = df.groupby(['database', 'cache'])['response_time_ms'].sem().unstack() #
std error of mean

x = range(len(cache_levels))
colors = {'PostgreSQL': 'blue', 'MongoDB': 'red'}
for i, db in enumerate(means.index):
    ax.errorbar(x, means.loc[db], yerr=errors.loc[db],
                marker='o', capsize=5, label=db, color=colors[db], linewidth=2)
ax.set_xticks(x)
ax.set_xticklabels(cache_levels)
ax.set_xlabel('Уровень кэширования')
ax.set_ylabel('Среднее время ответа (мс)')
ax.set_title('График взаимодействия с 95% доверительными интервалами')
ax.legend()
ax.grid(True, linestyle='--', alpha=0.3)
plt.show()

```

Шаг 6. Оценка размера эффекта (частичное η^2)

```

python
print("\n=== РАЗМЕР ЭФФЕКТА (Partial Eta-Squared) ===")

# Partial  $\eta^2 = SS_{effect} / (SS_{effect} + SS_{residual})$ 

```

```

ss_residual = anova_table['sum_sq']['Residual']

for effect in ['C(database)', 'C(cache)', 'C(database):C(cache)']:
    ss_effect = anova_table['sum_sq'][effect]
    partial_eta = ss_effect / (ss_effect + ss_residual)
    print(f"{effect}: partial  $\eta^2$  = {partial_eta:.4f}")
    if partial_eta < 0.01:
        print(f" → очень маленький эффект")
    elif partial_eta < 0.06:
        print(f" → маленький эффект")
    elif partial_eta < 0.14:
        print(f" → средний эффект")
    else:
        print(f" → большой эффект")

```

Шаг 7. Альтернатива: модель без взаимодействия (для сравнения)

```

python
print("\n" + "="*60)
print("МОДЕЛЬ БЕЗ ВЗАИМОДЕЙСТВИЯ (только главные эффекты)")
print("="*60)

model_additive = ols('response_time_ms ~ C(database) + C(cache)', data=df).fit()
anova_additive = sm.stats.anova_lm(model_additive, typ=2)
print(anova_additive)

# Сравнение AIC (меньше - лучше)
print(f"\nAIC модели с взаимодействием: {model_full.aic:.2f}")
print(f"AIC модели без взаимодействия: {model_additive.aic:.2f}")
if model_full.aic < model_additive.aic:
    print("✅ Модель с взаимодействием лучше описывает данные")
else:
    print("Модель без взаимодействия предпочтительнее")

```

Шаг 8. Итоговый вывод и рекомендации

```

python
print("\n" + "="*60)
print("ИТОГОВЫЙ ВЫВОД И РЕКОМЕНДАЦИИ")
print("="*60)

# Лучшая комбинация (минимальное среднее)
mean_by_combo = df.groupby(['database', 'cache'])['response_time_ms'].mean()

```

```

best_combo = mean_by_combo.idxmin()
best_value = mean_by_combo.min()

print(f"Лучшая конфигурация: {best_combo[0]} + {best_combo[1]}")
print(f"Среднее время ответа: {best_value:.1f} мс")

print("\nРекомендации:")
if p_interaction < alpha:
    print("• Обнаружено статистически значимое взаимодействие факторов.")
    print("• Эффективность кэширования зависит от типа базы данных.")
    if p_db < alpha:
        print("• Тип БД значимо влияет на производительность.")
    if p_cache < alpha:
        print("• Уровень кэширования значимо влияет на производительность.")
    print(f"• Рекомендуемая конфигурация: {best_combo[0]} с кэшем {best_combo[1]}")
else:
    print("• Взаимодействие факторов не обнаружено (эффекты аддитивны).")
    print("• Рекомендации можно давать независимо по каждому фактору.")

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов**. Необходимо выполнить полный анализ по шагам 1–8.

Вар.	Фактор А	Фактор В	Отклик	Характер взаимодействия
1	СУБД (PostgreSQL, MySQL)	Уровень кэша (none, Redis)	Время SELECT запроса (мс)	Сильное: PostgreSQL выигрывает больше
2	Алгоритм сжатия (gzip, brotli)	Размер блока (4KB, 64KB, 1MB)	Время сжатия (мс)	Слабое: для больших блоков brotli лучше
3	Сборщик GC (G1, ZGC)	Размер кучи (2GB, 8GB, 16GB)	Пауза GC (мс)	Пересечение: ZGC лучше при больших кучах
4	Сеть (WiFi, Ethernet)	Протокол (HTTP/1.1, HTTP/2)	Задержка (мс)	Отсутствует: эффекты аддитивны

Вар.	Фактор А	Фактор В	Отклик	Характер взаимодействия
5	ОС (Linux, Windows)	Файловая система (ext4, NTFS, APFS)	Время чтения файла (мс)	Умеренное: разное для разных ОС
6	Балансировщик (Nginx, HAProxy)	Алгоритм (round-robin, least-conn)	Время ответа при 1000 RPS (мс)	Есть: HAProxy лучше при least-conn

Генерация данных для варианта 1 (пример):

```
python
# PostgreSQL + no cache: 120, PostgreSQL + Redis: 60
# MySQL + no cache: 140, MySQL + Redis: 90
mean_vals = {
    ('PostgreSQL', 'no cache'): 120, ('PostgreSQL', 'Redis'): 60,
    ('MySQL', 'no cache'): 140, ('MySQL', 'Redis'): 90
}
```

7. Структура отчета

1. **Цель работы**
2. **Описание факторов, отклика и экспериментального дизайна**
3. **Визуальный анализ** (box-plot, график взаимодействий)
4. **Проверка предпосылок** (нормальность остатков, гомогенность дисперсий)
5. **Результаты двухфакторного ANOVA** (таблица, p-value, интерпретация)
6. **Анализ взаимодействия** (если значимо – простые эффекты, post-hoc)
7. **Размер эффекта** (partial η^2)
8. **Вывод и рекомендации** (лучшая конфигурация)
9. **Ответы на контрольные вопросы**
10. **Код**

8. Контрольные вопросы (для защиты)

1. Чем двухфакторный ANOVA отличается от двух отдельных однофакторных ANOVA?
2. Что такое эффект взаимодействия факторов? Приведите ИТ-пример.
3. Как по графику взаимодействия определить наличие взаимодействия?
4. Что означает значимое взаимодействие при незначимых главных эффектах?
5. Какой тип суммы квадратов (Type I, II, III) используется в `anova_lm` с `typ=2` и почему?

6. Что такое η^2 и чем он отличается от обычного η^2 ?
7. Почему важно проверять остатки на нормальность и гомогенность дисперсий?
8. Что делать, если взаимодействие значимо? Как интерпретировать простые эффекты?
9. В чём разница между сбалансированным и несбалансированным дизайном?
10. Какой непараметрический аналог двухфакторного ANOVA можно использовать?

9. Полный код в одном блоке (для копирования)

```
python
# Полный код двухфакторного ANOVA

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Генерация данных (пример: БД × Кэш)
np.random.seed(42)
data = []
for db, cache, mu in [('PG', 'NoCache', 150), ('PG', 'Redis', 70), ('PG', 'Mem', 80),
                      ('MG', 'NoCache', 200), ('MG', 'Redis', 120), ('MG', 'Mem', 130)]:
    for _ in range(10):
        val = np.random.normal(mu, 12)
        data.append({'db': db, 'cache': cache, 'time': val})
df = pd.DataFrame(data)

# Визуализация
sns.boxplot(data=df, x='cache', y='time', hue='db')
plt.show()

# График взаимодействий
pivot = df.pivot_table(index='db', columns='cache', values='time', aggfunc='mean')
pivot.T.plot(marker='o')
plt.title('Interaction plot')
plt.show()

# Two-way ANOVA
model = ols('time ~ C(db) * C(cache)', data=df).fit()
```

```
print(sm.stats.anova_lm(model, typ=2))
```

Данная лабораторная работа даёт практические навыки анализа сложных систем, где факторы влияют не независимо, а совместно – что является нормой в реальной программной инженерии.

ЛАБОРАТОРНАЯ РАБОТА №5

Тема: Построение множественной линейной регрессии (Multiple Linear Regression)

Время выполнения: 4 академических часа

Цель: освоить метод множественной линейной регрессии для прогнозирования отклика по нескольким факторам, научиться оценивать качество модели, проверять предпосылки и интерпретировать коэффициенты.

1. Актуальность для программной инженерии

В разработке ПО часто требуется **прогнозировать** одну метрику на основе нескольких других:

- **Прогнозирование времени выполнения ETL-процесса** на основе объёма данных (ГБ), числа потоков и уровня сжатия.
- **Оценка времени отклика API** в зависимости от количества запросов в секунду (RPS), размера полезной нагрузки (КБ) и загрузки CPU.
- **Предсказание стоимости облачного инстанса** на основе vCPU, RAM (ГБ), типа диска и региона.
- **Оценка времени сборки проекта** на основе числа файлов, строк кода и уровня оптимизации компилятора.

Множественная линейная регрессия позволяет:

1. Выявить **вклад каждого фактора** в изменение отклика.
2. Строить **прогностические модели** для новых данных.
3. Проверять статистическую значимость факторов.
4. Оценивать качество модели (R^2 , скорректированный R^2 , AIC, BIC).

2. Теоретическая справка (множественная линейная регрессия)

Модель:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon$$

Где:

- Y – отклик (зависимая переменная)
- $X_1, X_2, \dots, X_p$ – факторы (независимые переменные)
- β_0 – свободный член (intercept)
- $\beta_1, \dots, \beta_p$ – коэффициенты регрессии

- $\varepsilon\varepsilon$ – случайная ошибка

Метод оценки коэффициентов: метод наименьших квадратов (OLS – Ordinary Least Squares).

Качество модели:

Метрика	Формула	Интерпретация
R² (коэффициент детерминации)	$1 - \frac{SS_{res}}{SS_{tot}}$	Доля дисперсии Y, объяснённая моделью (0–1)
Adjusted R²	$1 - \frac{(1 - R^2)(n - 1)}{n - p - 1}$	R ² с поправкой на число факторов
AIC / BIC	–	Чем меньше, тем лучше (для сравнения моделей)
RMSE	$\sqrt{\frac{1}{n} \sum (y_i - \hat{y}_i)^2}$	Среднеквадратичная ошибка прогноза

Предпосылки классической линейной регрессии (Gauss-Markov):

1. **Линейность** – связь между Y и X линейна.
2. **Независимость ошибок** – остатки не коррелируют.
3. **Гомоскедастичность** – постоянная дисперсия остатков.
4. **Нормальность остатков** – для корректных доверительных интервалов.
5. **Отсутствие мультиколлинеарности** – факторы не должны быть сильно скоррелированы (VIF < 5–10).

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – инженер по данным (Data Engineer). Ваш ETL-процесс обрабатывает ежедневные логи. Вы собрали данные о **40 запусках** процесса.

Факторы (независимые переменные):

Фактор	Описание	Диапазон
data_volume_gb	Объём входных данных (ГБ)	10–100
num_threads	Количество потоков обработки	1–8

Фактор	Описание	Диапазон
compression_level	Уровень сжатия (0 – нет, 9 – максимум)	0–9
cpu_load	Фоновая загрузка CPU (%)	5–80

Отклик (зависимая переменная):

etl_time_min – время выполнения ETL (в минутах).

Задача:

1. Провести разведочный анализ и корреляционный анализ.
2. Построить множественную линейную регрессию.
3. Оценить качество модели (R^2 , adjusted R^2 , RMSE, AIC).
4. Проверить предпосылки регрессии (остатки, мультиколлинеарность).
5. Интерпретировать коэффициенты (вклад каждого фактора).
6. Сделать прогноз для новой конфигурации.
7. Дать рекомендации по оптимизации ETL.

4. Исходные данные (генерация)

Создайте датасет с реалистичной зависимостью.

```
python
import numpy as np
import pandas as pd

np.random.seed(42)
n = 40

# Генерация факторов
data_volume_gb = np.random.uniform(10, 100, n)
num_threads = np.random.randint(1, 9, n)
compression_level = np.random.randint(0, 10, n)
cpu_load = np.random.uniform(5, 80, n)

# Формула генерации времени ETL (с шумом)
# Базовое время: 5 + 0.8*volume + 0.5*compression - 1.2*threads + 0.05*cpu_load
etl_time_min = (5.0 +
                0.8 * data_volume_gb +
                0.5 * compression_level -
                1.2 * num_threads +
```

```

0.05 * cpu_load +
np.random.normal(0, 3, n)) # шум

# Добавляем нелинейность: снижение эффективности при threads > 6
etl_time_min = np.where(num_threads > 6, etl_time_min + 2.0, etl_time_min)

# Округляем и гарантируем положительность
etl_time_min = np.maximum(etl_time_min, 1)

df = pd.DataFrame({
    'data_volume_gb': data_volume_gb,
    'num_threads': num_threads,
    'compression_level': compression_level,
    'cpu_load': cpu_load,
    'etl_time_min': etl_time_min
})

print("Первые 10 строк:")
print(df.head(10))
print("\nОписательные статистики:")
print(df.describe())

```

5. Ход выполнения (пошаговое руководство)

Работа выполняется в **Jupyter Notebook** или **Google Colab**.

Шаг 1. Импорт библиотек

```

python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.diagnostic import het_breuschpagan
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')

```

```
sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)
```

Шаг 2. Разведочный анализ и корреляции

```
python
# Матрица корреляций
corr_matrix = df.corr()
print("=== Матрица корреляций ===")
print(corr_matrix.round(3))

# Тепловая карта корреляций
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0, fmt='.2f')
plt.title('Корреляционная матрица')
plt.show()

# Парные графики
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
features = ['data_volume_gb', 'num_threads', 'compression_level', 'cpu_load']
for i, feat in enumerate(features):
    row, col = i // 2, i % 2
    axes[row, col].scatter(df[feat], df['etl_time_min'], alpha=0.7)
    axes[row, col].set_xlabel(feat)
    axes[row, col].set_ylabel('etl_time_min')
    axes[row, col].set_title(f'Зависимость от {feat}')
    # Линия тренда
    z = np.polyfit(df[feat], df['etl_time_min'], 1)
    p = np.poly1d(z)
    axes[row, col].plot(df[feat].sort_values(), p(df[feat].sort_values()), "r--", alp
ha=0.8)
plt.tight_layout()
plt.show()
```

Вопрос: Какие факторы имеют наиболее сильную корреляцию с откликом? Есть ли мультиколлинеарность?

Шаг 3. Построение множественной регрессии

```
python
print("="*60)
print("МНОЖЕСТВЕННАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ")
print("="*60)
```

```

# Определяем X и y
X = df[['data_volume_gb', 'num_threads', 'compression_level', 'cpu_load']]
y = df['etl_time_min']

# Добавляем константу
X_with_const = sm.add_constant(X)

# Оценка модели
model = sm.OLS(y, X_with_const).fit()
print(model.summary())

# Ключевые метрики
print(f"\n=== КЛЮЧЕВЫЕ МЕТРИКИ ===")
print(f"R² = {model.rsquared:.4f}")
print(f"Adj. R² = {model.rsquared_adj:.4f}")
print(f"F-statistic = {model.fvalue:.2f} (p = {model.f_pvalue:.4e})")
print(f"AIC = {model.aic:.2f}")
print(f"BIC = {model.bic:.2f}")
print(f"RMSE = {np.sqrt(model.mse_resid):.3f} мин")

```

Шаг 4. Интерпретация коэффициентов

```

python
print("\n=== КОЭФФИЦИЕНТЫ РЕГРЕССИИ ===")
coef_df = pd.DataFrame({
    'Фактор': model.params.index,
    'Коэффициент': model.params.values,
    'Стандартная ошибка': model.bse.values,
    't-статистика': model.tvalues.values,
    'p-value': model.pvalues.values,
    'Нижний 95% CI': model.conf_int()[0],
    'Верхний 95% CI': model.conf_int()[1]
})
print(coef_df.round(4))

print("\n=== ИНТЕРПРЕТАЦИЯ ===")
for var, coef in zip(model.params.index[1:], model.params.values[1:]):
    if coef > 0:
        print(f"• Увеличение {var} на 1 единицу увеличивает etl_time_min на {coef:.3f} мин")
    else:

```

```
print(f"• Увеличение {var} на 1 единицу уменьшает etl_time_min на {abs(coef):.3f} мин")
```

```
print(f"\n• Константа (const = {model.params['const']:.3f}): базовое время при нулевых факторах")
```

Шаг 5. Проверка мультиколлинеарности (VIF)

```
python
print("\n=== ПРОВЕРКА МУЛЬТИКОЛЛИНЕАРНОСТИ (VIF) ===")
vif_data = pd.DataFrame()
vif_data['Фактор'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif_data.round(2))

if (vif_data['VIF'] > 5).any():
    print("\n Высокий VIF (>5) указывает на мультиколлинеарность.")
    print("Рекомендуется удалить один из коррелированных факторов.")
else:
    print("\n✅Мультиколлинеарность отсутствует (VIF < 5).")
```

Шаг 6. Анализ остатков (предпосылки)

```
python
residuals = model.resid
fitted = model.fittedvalues

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# 1. Остатки vs Предсказанные значения (гомоскедастичность)
axes[0,0].scatter(fitted, residuals, alpha=0.7)
axes[0,0].axhline(y=0, color='red', linestyle='--')
axes[0,0].set_xlabel('Предсказанные значения')
axes[0,0].set_ylabel('Остатки')
axes[0,0].set_title('Остатки vs Предсказанные значения')

# 2. Q-Q plot (нормальность остатков)
stats.probplot(residuals, dist="norm", plot=axes[0,1])
axes[0,1].set_title('Q-Q plot остатков')

# 3. Гистограмма остатков
axes[1,0].hist(residuals, bins=15, edgecolor='black', alpha=0.7)
axes[1,0].set_xlabel('Остатки')
```

```

axes[1,0].set_ylabel('Частота')
axes[1,0].set_title('Гистограмма остатков')

# 4. Тест на нормальность (Shapiro-Wilk)
shapiro_stat, shapiro_p = stats.shapiro(residuals)
axes[1,1].text(0.1, 0.5, f'Shapiro-Wilk test:\np-value = {shapiro_p:.4f}',
               fontsize=14, transform=axes[1,1].transAxes)
axes[1,1].axis('off')
if shapiro_p > 0.05:
    axes[1,1].text(0.1, 0.3, '☑Остатки нормальны', fontsize=14, color='green', trans
form=axes[1,1].transAxes)
else:
    axes[1,1].text(0.1, 0.3, '☐ Остатки не нормальны', fontsize=14, color='red', tra
nsform=axes[1,1].transAxes)

plt.tight_layout()
plt.show()

# Тест Бройша-Пагана (гомоскедастичность)
bp_stat, bp_p, _, _ = het_breuschpagan(residuals, X_with_const)
print(f"\nТест Бройша-Пагана (гомоскедастичность): p-value = {bp_p:.4f}")
if bp_p > 0.05:
    print("☑Дисперсия остатков постоянна (гомоскедастичность)")
else:
    print("☐ Обнаружена гетероскедастичность (дисперсия непостоянна)")

```

Шаг 7. Прогнозирование для новых данных

```

python
print("\n=== ПРОГНОЗИРОВАНИЕ ===")

# Новая конфигурация
new_data = pd.DataFrame({
    'data_volume_gb': [50, 75],
    'num_threads': [4, 8],
    'compression_level': [3, 6],
    'cpu_load': [20, 45]
})

# Добавляем константу
new_data_const = sm.add_constant(new_data)

# Прогноз

```

```

predictions = model.predict(new_data_const)
prediction_ci = model.get_prediction(new_data_const).conf_int()

print("Прогноз времени ETL для новых конфигураций:")
for i in range(len(new_data)):
    print(f"\nКонфигурация {i+1}:")
    print(f"    Объём={new_data['data_volume_gb'].iloc[i]} ГБ, потоки={new_data['num_th
reads'].iloc[i]}, "
          f"сжатие={new_data['compression_level'].iloc[i]}, CPU={new_data['cpu_load']
.iloc[i]}%")
    print(f"    Прогноз: {predictions[i]:.1f} мин")
    print(f"    95% ДИ: [{prediction_ci[i][0]:.1f}, {prediction_ci[i][1]:.1f}] мин")

```

Шаг 8. Визуализация качества прогноза

```

python
# Фактические vs Предсказанные значения
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# График фактических vs предсказанных
axes[0].scatter(y, fitted, alpha=0.7)
axes[0].plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
axes[0].set_xlabel('Фактические значения (мин)')
axes[0].set_ylabel('Предсказанные значения (мин)')
axes[0].set_title(f'Фактические vs Предсказанные (R² = {model.rsquared:.3f})')

# Остатки по порядку наблюдений
axes[1].plot(residuals, 'o-', alpha=0.7)
axes[1].axhline(y=0, color='red', linestyle='--')
axes[1].set_xlabel('Номер наблюдения')
axes[1].set_ylabel('Остатки')
axes[1].set_title('Остатки по порядку наблюдений')
axes[1].grid(True, linestyle='--', alpha=0.3)

plt.tight_layout()
plt.show()

```

Шаг 9. Итоговый вывод и рекомендации

```

python
print("\n" + "="*60)
print("ИТОГОВЫЙ ВЫВОД И РЕКОМЕНДАЦИИ")
print("="*60)

```

```

print(f"Модель объясняет {model.rsquared*100:.1f}% дисперсии времени ETL.")

# Факторы с p-value < 0.05
significant = model.pvalues[model.pvalues < 0.05].index.tolist()
print(f"\nСтатистически значимые факторы (p < 0.05): {significant}")

print("\nРекомендации по оптимизации ETL:")
# Находим факторы с отрицательным коэффициентом (ускоряют)
for var, coef, pval in zip(model.params.index[1:], model.params.values[1:], model.pvalues.values[1:]):
    if coef < 0 and pval < 0.05:
        print(f" • Увеличение {var} сокращает время. Рекомендуется увеличить {var}."
        )
    elif coef > 0 and pval < 0.05:
        print(f" • Снижение {var} сокращает время. Рекомендуется уменьшить {var}.")

# Проверка насыщения для потоков
max_threads_effective = 6
print(f"\n • Эффективность от увеличения потоков снижается после {max_threads_effective} потоков.")
print(f" • Рекомендуемое значение num_threads: {max_threads_effective}")

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов**. Необходимо выполнить полный анализ по шагам 1–9.

Вар.	Отклик	Факторы	Контекст
1	Время ответа API (мс)	RPS, размер payload (КБ), число инстансов	Оптимизация веб-сервиса
2	Время сборки (сек)	Число файлов, строк кода (тыс), уровень оптимизации (-O0..-O3)	CI/CD оптимизация
3	Потребление памяти (МБ)	Размер входных данных (МБ), число потоков, версия алгоритма	Анализ алгоритма

Вар.	Отклик	Факторы	Контекст
4	Стоимость облака (\$/мес)	vCPU, RAM (ГБ), трафик (ТБ), регион	FinOps
5	Частота ошибок (%)	Число запросов, время ответа (мс), загрузка CPU	SRE
6	Энергопотребление (Вт)	Частота CPU (ГГц), число ядер, тип нагрузки	Green IT

7. Структура отчета

1. **Цель работы**
2. **Описание данных** (факторы, отклик, объём выборки)
3. **Разведочный анализ** (корреляции, парные графики)
4. **Модель регрессии** (коэффициенты, их интерпретация)
5. **Качество модели** (R^2 , adj. R^2 , RMSE, AIC, F-тест)
6. **Проверка предпосылок** (остатки, мультиколлинеарность)
7. **Прогнозирование** для новых данных
8. **Вывод и рекомендации**
9. **Ответы на контрольные вопросы**
10. **Код** (ссылка или вставка)

8. Контрольные вопросы (для защиты)

1. Что означает коэффициент детерминации R^2 ? Чем adjusted R^2 отличается от обычного R^2 ?
2. Как интерпретировать коэффициент регрессии β_j при факторе X_j ?
3. Что показывает p-value для коэффициента регрессии?
4. Что такое мультиколлинеарность и как её обнаружить (VIF)?
5. Какие предпосылки должны выполняться для корректности выводов линейной регрессии?
6. Как проверить гомоскедастичность остатков?
7. Зачем нужны доверительные интервалы для прогнозов?
8. Что делать, если фактор имеет p-value > 0.05?
9. В чём разница между train/test split и использованием всей выборки?
10. Какой метод используется для оценки коэффициентов в линейной регрессии?

9. Полный код в одном блоке (для копирования)

```
python
# Полный код множественной линейной регрессии

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Генерация данных
np.random.seed(42)
n = 40
df = pd.DataFrame({
    'volume': np.random.uniform(10, 100, n),
    'threads': np.random.randint(1, 9, n),
    'compress': np.random.randint(0, 10, n),
    'cpu': np.random.uniform(5, 80, n)
})
df['time'] = (5 + 0.8*df['volume'] + 0.5*df['compress'] - 1.2*df['threads'] + 0.05*df
['cpu'] + np.random.normal(0, 3, n))
df.loc[df['threads'] > 6, 'time'] += 2

# Модель
X = sm.add_constant(df[['volume', 'threads', 'compress', 'cpu']])
model = sm.OLS(df['time'], X).fit()
print(model.summary())

# VIF
vif = [variance_inflation_factor(df[['volume', 'threads', 'compress', 'cpu']].values,
i) for i in range(4)]
print("\nVIF:", dict(zip(['volume', 'threads', 'compress', 'cpu'], [round(v, 2) for v in
vif])))

# Прогноз
new = pd.DataFrame({'const':1, 'volume':[50], 'threads':[4], 'compress':[3], 'cpu':[2
0]})
pred = model.predict(new)
print(f"\nПрогноз для volume=50, threads=4, compress=3, cpu=20: {pred.iloc[0]:.1f} ми
н")
```

Данная лабораторная работа даёт практические навыки построения прогностических моделей, которые широко используются в Data Science, Site Reliability Engineering (SRE) и оптимизации производительности программных систем.

ЛАБОРАТОРНАЯ РАБОТА №6

Тема: Полный факторный эксперимент (ПФЭ 2^k)

Время выполнения: 4 академических часа

Цель: освоить методологию активного планирования эксперимента для выявления влияния нескольких факторов и их взаимодействий на отклик, научиться строить матрицу планирования, оценивать эффекты и оптимизировать параметры.

1. Актуальность для программной инженерии

В отличие от пассивного наблюдения (регрессионного анализа), **активный эксперимент** позволяет целенаправленно изменять факторы для получения максимальной информации при минимальном числе опытов.

ИТ-примеры применения ПФЭ 2^k :

Факторы	Отклик	Цель
Размер кэша, число потоков, тип диска	QPS (запросов/сек)	Найти оптимальную конфигурацию сервера
Уровень сжатия, размер блока, тип алгоритма	Время сжатия и размер результата	Баланс между скоростью и степенью сжатия
Таймауты, буфера, keepralive	Задержка и пропускная способность	Настройка балансировщика нагрузки
Параметры JVM (Xms, Xmx, GC)	Время паузы GC	Тонкая настройка Java-приложения

Преимущества ПФЭ:

- Минимальное число опытов: для k факторов – 2^k опытов.
- Позволяет оценить **все главные эффекты** и **взаимодействия** (до порядка k).
- Ортогональность матрицы планирования → независимые оценки эффектов.

2. Теоретическая справка (ПФЭ 2^k)

Основные понятия:

Термин	Описание
Фактор	Входная переменная, влияющая на отклик
Уровень фактора	Значение фактора в эксперименте. В ПФЭ 2^k – два уровня: нижний (-1) и верхний (+1)
Отклик	Выходная переменная (результат опыта)
Матрица планирования	Таблица, в которой строки – опыты, столбцы – факторы (в кодированных значениях)
Эффект фактора	Изменение отклика при переходе фактора с -1 на +1: $E = \frac{\sum y_{(x_i=+1)}}{n/2} - \frac{\sum y_{(x_i=-1)}}{n/2}$
Взаимодействие	Эффект совместного изменения двух и более факторов. Обозначается произведением (A×B)

Матрица планирования для ПФЭ 2^3 :

Опыт	A	B	C	A×B	A×C	B×C	A×B×C	Отклик y
1	-1	-1	-1	+1	+1	+1	-1	y ₁
2	+1	-1	-1	-1	-1	+1	+1	y ₂
3	-1	+1	-1	-1	+1	-1	+1	y ₃
4	+1	+1	-1	+1	-1	-1	-1	y ₄
5	-1	-1	+1	+1	-1	-1	+1	y ₅
6	+1	-1	+1	-1	+1	-1	-1	y ₆
7	-1	+1	+1	-1	-1	+1	-1	y ₇
8	+1	+1	+1	+1	+1	+1	+1	y ₈

Кодирование натуральных значений:

$$x = \frac{X - X_0}{\Delta X / 2}, \quad \text{где } X_0 = \frac{X_{\max} + X_{\min}}{2}, \quad \Delta X = X_{\max} - X_{\min}$$

Регрессионная модель:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3 + b_{123}x_1x_2x_3$$

Оценка коэффициентов:

$$b_0 = \frac{\sum y}{n}, \quad b_j = \frac{\sum x_{ij}y_i}{n}, \quad b_{jl} = \frac{\sum (x_{ij}x_{il})y_i}{n}$$

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – инженер по производительности. Ваш веб-сервер (Nginx) обрабатывает статические файлы. Вы хотите найти оптимальную конфигурацию для максимизации **пропускной способности (RPS – requests per second)**.

Факторы (k=3, по два уровня):

Фактор	Обозначение	Нижний уровень (-1)	Верхний уровень (+1)
Размер кэша (МБ)	A	64	512
Число рабочих процессов	B	2	8
Keepalive timeout (сек)	C	10	120

Отклик:

rps – количество запросов в секунду (измеряется нагрузочным тестированием).

Задача:

1. Построить матрицу планирования ПФЭ 2^3 .
2. Провести эксперимент (симуляция или реальные замеры).
3. Рассчитать главные эффекты и эффекты взаимодействий.
4. Построить регрессионную модель.
5. Определить статистически значимые эффекты (t-критерий).

6. Выполнить оптимизацию (определить направление движения к оптимуму).
7. Дать рекомендации по настройке сервера.

4. Исходные данные (генерация откликов)

Создайте функцию, моделирующую поведение RPS в зависимости от факторов (с взаимодействиями).

```
python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from itertools import product
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)

def simulate_rps(cache_mb, workers, keepalive):
    """
    Симуляция пропускной способности веб-сервера.
    cache_mb: 64..512
    workers: 2..8
    keepalive: 10..120
    """
    # Нормируем факторы к диапазону [0,1] для модели
    cache_norm = (cache_mb - 64) / (512 - 64)
    workers_norm = (workers - 2) / (8 - 2)
    keepalive_norm = (keepalive - 10) / (120 - 10)

    # Истинная модель (с взаимодействиями)
    base = 1000
    effect_cache = 800 * cache_norm
    effect_workers = 600 * workers_norm
    effect_keepalive = 300 * keepalive_norm
    interaction_cache_workers = 400 * cache_norm * workers_norm # положительное
    interaction_workers_keepalive = -200 * workers_norm * keepalive_norm # отрицательное

    rps = (base + effect_cache + effect_workers + effect_keepalive +
```

```

        interaction_cache_workers + interaction_workers_keepalive +
        np.random.normal(0, 30)) # шум

    return max(0, int(rps))

# Данные для ПФЭ 2^3
cache_levels = {'-1': 64, '+1': 512}
workers_levels = {'-1': 2, '+1': 8}
keepalive_levels = {'-1': 10, '+1': 120}

# Матрица планирования
design_matrix = []
for cache, workers, keepalive in product([-1, 1], repeat=3):
    cache_val = cache_levels[str(cache)]
    workers_val = workers_levels[str(workers)]
    keepalive_val = keepalive_levels[str(keepalive)]

    rps = simulate_rps(cache_val, workers_val, keepalive_val)

    design_matrix.append({
        'A (cache)': cache,
        'B (workers)': workers,
        'C (keepalive)': keepalive,
        'A_val': cache_val,
        'B_val': workers_val,
        'C_val': keepalive_val,
        'RPS': rps
    })

df = pd.DataFrame(design_matrix)
print("=== Матрица планирования ПФЭ 2^3 ===")
print(df[['A (cache)', 'B (workers)', 'C (keepalive)', 'A_val', 'B_val', 'C_val', 'RPS']].to_string())
print(f"\nСредний RPS: {df['RPS'].mean():.0f}")

```

5. Ход выполнения (пошаговое руководство)

Шаг 1. Импорт библиотек и подготовка данных

```

python
import numpy as np
import pandas as pd

```

```

import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from statsmodels.formula.api import ols
from itertools import product

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 5)

# Проверка сбалансированности
print(f"Количество опытов: {len(df)}")
print(f"Ожидаемое количество:  $2^3 = 8$ ")

```

Шаг 2. Добавление столбцов взаимодействий

```

python
# Добавляем столбцы для взаимодействий
df['AB'] = df['A (cache)'] * df['B (workers)']
df['AC'] = df['A (cache)'] * df['C (keepalive)']
df['BC'] = df['B (workers)'] * df['C (keepalive)']
df['ABC'] = df['A (cache)'] * df['B (workers)'] * df['C (keepalive)']

print("\n=== Полная матрица планирования с взаимодействиями ===")
print(df[['A (cache)', 'B (workers)', 'C (keepalive)', 'AB', 'AC', 'BC', 'ABC', 'RPS'
]].to_string())

```

Шаг 3. Расчёт эффектов факторов и взаимодействий

```

python
print("\n" + "="*60)
print("РАСЧЁТ ЭФФЕКТОВ (ПФЭ  $2^3$ )")
print("="*60)

n = len(df) # 8
y = df['RPS'].values

# Список всех эффектов
effects = {
    'A (cache)': df['A (cache)'].values,
    'B (workers)': df['B (workers)'].values,
    'C (keepalive)': df['C (keepalive)'].values,
    'AB': df['AB'].values,
    'AC': df['AC'].values,

```

```

    'BC': df['BC'].values,
    'ABC': df['ABC'].values
}

# Расчёт эффекта по формуле:  $E = (1/n) * \sum (x_i * y_i) * 2$ ?
# Для ПФЭ  $2^k$ : эффект = (сумма у при +1) / (n/2) - (сумма у при -1) / (n/2)
# Или проще: effect = (2/n) *  $\sum (x_i * y_i)$ 

effect_values = {}
for name, x in effects.items():
    # Классическая формула: эффект = (среднее при +1) - (среднее при -1)
    y_plus = y[x == 1].mean()
    y_minus = y[x == -1].mean()
    effect = y_plus - y_minus
    effect_values[name] = effect

# Коэффициенты регрессии  $b = effect / 2$ 
coefficients = {name: val / 2 for name, val in effect_values.items()}
coefficients['const'] = y.mean()

# Вывод результатов
results_df = pd.DataFrame({
    'Эффект': effect_values.keys(),
    'Значение эффекта': effect_values.values(),
    'Коэффициент b': coefficients.values() if len(coefficients) == len(effects) else
    list(coefficients.values())[:len(effects)]
})

# Добавим константу отдельно
print("\n=== Эффекты факторов и взаимодействий ===")
for name, eff in effect_values.items():
    print(f"{name:15} : {eff:8.2f} RPS")

print(f"\n=== Коэффициенты регрессии (b = effect/2) ===")
print(f"b0 (const) : {y.mean():8.2f} RPS")
for name, coef in coefficients.items():
    print(f"b_{name:12} : {coef:8.2f} RPS")

```

Шаг 4. Визуализация эффектов (Pareto chart)

```

python
# Сортировка эффектов по абсолютной величине
effects_sorted = sorted(effect_values.items(), key=lambda x: abs(x[1]), reverse=True)

```

```

fig, ax = plt.subplots(figsize=(10, 6))
names = [e[0] for e in effects_sorted]
values = [e[1] for e in effects_sorted]
colors = ['red' if v < 0 else 'green' for v in values]

bars = ax.barh(names, values, color=colors)
ax.axvline(x=0, color='black', linewidth=0.8)
ax.set_xlabel('Величина эффекта (RPS)')
ax.set_title('Диаграмма Парето эффектов (ПФЭ 2³)')
ax.grid(axis='x', linestyle='--', alpha=0.7)

# Добавляем подписи
for i, (name, val) in enumerate(effects_sorted):
    ax.text(val + (5 if val > 0 else -35), i, f'{val:.0f}', va='center')

plt.tight_layout()
plt.show()

```

Шаг 5. Статистическая значимость эффектов (t-критерий)

```

python
print("\n=== ОЦЕНКА СТАТИСТИЧЕСКОЙ ЗНАЧИМОСТИ ===")

# Оценка дисперсии воспроизводимости (нужны параллельные опыты)
# Для демонстрации используем остатки от модели

# Строим модель
X = df[['A (cache)', 'B (workers)', 'C (keepalive)', 'AB', 'AC', 'BC', 'ABC']].copy()
X = sm.add_constant(X)
model = sm.OLS(df['RPS'], X).fit()

# Получаем p-value
p_values = model.pvalues
print("\np-value для эффектов (H0: эффект = 0):")
for name, p in p_values.items():
    if name == 'const':
        continue
    sign = "✓значим" if p < 0.05 else "✗не значим"
    print(f" {name:15} : p = {p:.4f} {sign}")

# Критический уровень эффекта (при отсутствии параллельных опытов)
# Используем t-критерий: |effect| > t_{0.05, df} * sqrt(2 * s^2 / n)
# При df = 0 (нет повторений) – только качественная оценка

```

```
print("\n□ Для точной оценки значимости необходимы параллельные опыты.")
print("    Рекомендуется провести 2-3 повторения каждого опыта.")
```

Шаг 6. Построение регрессионной модели

```
python
print("\n" + "="*60)
print("РЕГРЕССИОННАЯ МОДЕЛЬ В КОДИРОВАННЫХ ПЕРЕМЕННЫХ")
print("="*60)

# Формируем формулу
formula = 'RPS ~ A (cache) + B (workers) + C (keepalive) + AB + AC + BC + ABC'
model_ols = ols(formula, data=df).fit()
print(model_ols.summary())

# Уравнение регрессии
eq_terms = [f"{model_ols.params['const']:.1f}"]
for term in ['A (cache)', 'B (workers)', 'C (keepalive)', 'AB', 'AC', 'BC', 'ABC']:
    coef = model_ols.params[term]
    if coef > 0:
        eq_terms.append(f"+ {coef:.1f}·{term}")
    elif coef < 0:
        eq_terms.append(f"- {abs(coef):.1f}·{term}")

print(f"\nМодель: RPS = {' '.join(eq_terms)}")
```

Шаг 7. Графический анализ эффектов

```
python
# Графики главных эффектов
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

factors = ['A (cache)', 'B (workers)', 'C (keepalive)']
factor_names = ['Размер кэша', 'Число рабочих процессов', 'Keepalive timeout']

for i, (factor, name) in enumerate(zip(factors, factor_names)):
    # Средние при уровнях -1 и +1
    mean_minus = df[df[factor] == -1]['RPS'].mean()
    mean_plus = df[df[factor] == 1]['RPS'].mean()

    axes[i].bar(['-1 (низкий)', '+1 (высокий)'], [mean_minus, mean_plus],
                color=['steelblue', 'coral'])
    axes[i].set_ylabel('Средний RPS')
```

```

axes[i].set_title(f'Главный эффект: {name}')
axes[i].grid(axis='y', linestyle='--', alpha=0.7)

# Добавляем значение эффекта
effect = mean_plus - mean_minus
axes[i].text(0.5, min(mean_minus, mean_plus) - 20, f'Эффект = {effect:.0f}',
            ha='center', fontsize=11)

plt.tight_layout()
plt.show()

# Графики взаимодействий
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Взаимодействие AxB
pivot_ab = df.pivot_table(values='RPS', index='A (cache)', columns='B (workers)', agg
func='mean')
axes[0].plot([-1, 1], [pivot_ab.loc[-1, -1], pivot_ab.loc[1, -1]], 'o-', label='B = -
1 (workers=2)')
axes[0].plot([-1, 1], [pivot_ab.loc[-1, 1], pivot_ab.loc[1, 1]], 's-', label='B = +1
(workers=8)')
axes[0].set_xlabel('A (cache)')
axes[0].set_ylabel('Средний RPS')
axes[0].set_title('Взаимодействие AxB (кэш × workers)')
axes[0].legend()
axes[0].grid(True)

# Взаимодействие BxC
pivot_bc = df.pivot_table(values='RPS', index='B (workers)', columns='C (keepalive)',
aggfunc='mean')
axes[1].plot([-1, 1], [pivot_bc.loc[-1, -1], pivot_bc.loc[1, -1]], 'o-', label='C = -
1 (timeout=10)')
axes[1].plot([-1, 1], [pivot_bc.loc[-1, 1], pivot_bc.loc[1, 1]], 's-', label='C = +1
(timeout=120)')
axes[1].set_xlabel('B (workers)')
axes[1].set_ylabel('Средний RPS')
axes[1].set_title('Взаимодействие BxC (workers × keepalive)')
axes[1].legend()
axes[1].grid(True)

plt.tight_layout()
plt.show()

```

Шаг 8. Оптимизация и направление движения

```
python
print("\n" + "="*60)
print("ОПТИМИЗАЦИЯ И РЕКОМЕНДАЦИИ")
print("="*60)

# Определяем, какие факторы нужно увеличить/уменьшить
print("Направление движения к максимуму RPS:")
for factor in ['A (cache)', 'B (workers)', 'C (keepalive)']:
    effect = effect_values[factor]
    if effect > 0:
        print(f"    • {factor}: эффект положительный ({effect:.0f}) → УВЕЛИЧИТЬ")
    else:
        print(f"    • {factor}: эффект отрицательный ({effect:.0f}) → УМЕНЬШИТЬ")

# Лучшая комбинация из проведённых опытов
best_row = df.loc[df['RPS'].idxmax()]
print(f"\nЛучшая конфигурация из 8 опытов:")
print(f"  A (cache) = {best_row['A_val']} МБ")
print(f"  B (workers) = {best_row['B_val']}")
print(f"  C (keepalive) = {best_row['C_val']} сек")
print(f"  RPS = {best_row['RPS']:.0f}")

# Рекомендация по следующему эксперименту (крутое восхождение)
print(f"\nРекомендация для следующего эксперимента (крутое восхождение):")
print("  Установить факторы на верхних уровнях для положительных эффектов")
print("  и на нижних уровнях для отрицательных эффектов.")
print("  Провести ПФЭ с новыми базовыми уровнями.")
```

Шаг 9. Проверка адекватности модели

```
python
print("\n" + "="*60)
print("ПРОВЕРКА АДЕКВАТНОСТИ МОДЕЛИ")
print("="*60)

# Предсказанные значения
df['RPS_pred'] = model_ols.predict(df)

# Ошибка предсказания
df['residual'] = df['RPS'] - df['RPS_pred']
```

```

print("Сравнение фактических и предсказанных значений:")
print(df[['A (cache)', 'B (workers)', 'C (keepalive)', 'RPS', 'RPS_pred', 'residual']
].to_string())

# График фактических vs предсказанных
plt.figure(figsize=(8, 6))
plt.scatter(df['RPS'], df['RPS_pred'], s=100, alpha=0.7)
plt.plot([df['RPS'].min(), df['RPS'].max()],
         [df['RPS'].min(), df['RPS'].max()], 'r--', lw=2)
plt.xlabel('Фактический RPS')
plt.ylabel('Предсказанный RPS')
plt.title(f'Фактические vs Предсказанные значения ( $R^2 = \{model\_ols.rsquared:.3f\}$ )')
plt.grid(True, linestyle='--', alpha=0.3)
plt.show()

# Оценка адекватности: критерий Фишера (нужны параллельные опыты)
print("\n□ Для проверки адекватности модели необходимы параллельные опыты")
print("    в одной или нескольких точках факторного пространства.")

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов**. Необходимо выполнить полный анализ по шагам 1–9.

Вар.	Фактор А	Фактор В	Фактор С	Отклик	Контекст
1	Размер кэша (64/512 МБ)	Число потоков (2/8)	Keepalive (10/120 с)	RPS	Веб-сервер
2	Размер буфера (4/64 КБ)	Число соединений (100/1000)	Сжатие (off/on)	Задержка (мс)	Балансировщик
3	Размер блока (4/64 КБ)	Уровень сжатия (1/9)	Число потоков (1/8)	Время сжатия (с)	Алгоритм gzip
4	Xms	Xmx	GC тип	Пауза GC (мс)	JVM

Вар.	Фактор А	Фактор В	Фактор С	Отклик	Контекст
	(512/4096 МБ)	(1024/8192 МБ)	(G1/ZGC)		
5	Размер страницы (4/64 КБ)	Кэш БД (off/on)	Индекс (off/on)	Время запроса (мс)	PostgreSQL
6	Частота CPU (2.0/3.5 ГГц)	Число ядер (2/8)	Тип нагрузки (IO/CPU)	Энергопотребление (Вт)	Green IT

7. Структура отчета

1. **Цель работы**
2. **Описание факторов и отклика**
3. **Матрица планирования** (таблица с кодированными и натуральными значениями)
4. **Расчёт эффектов** (главные эффекты и взаимодействия)
5. **Регрессионная модель** (коэффициенты, уравнение)
6. **Статистическая значимость эффектов**
7. **Графический анализ** (Pareto chart, графики взаимодействий)
8. **Оптимизация** (лучшая конфигурация, направление движения)
9. **Выводы и рекомендации**
10. **Ответы на контрольные вопросы**

8. Контрольные вопросы (для защиты)

1. Почему в ПФЭ 2^k факторы кодируются значениями -1 и +1?
2. Как рассчитать главный эффект фактора по матрице планирования?
3. Чем отличается эффект фактора от коэффициента регрессии b ?
4. Как рассчитать эффект взаимодействия двух факторов?
5. Почему матрица ПФЭ 2^k называется ортогональной?
6. Что такое «рандомизация» и зачем она нужна в активном эксперименте?
7. Как проверить статистическую значимость эффектов при отсутствии параллельных опытов?
8. Что такое «смешивание эффектов» (confounding) и в каких планах оно возникает?
9. Как определить направление движения к оптимуму (крутое восхождение)?

10. Чем ПФЭ отличается от пассивного регрессионного анализа?

9. Полный код в одном блоке (для копирования)

```
python
# Полный код ПФЭ 2^3

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from itertools import product
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Симуляция отклика
def simulate_rps(cache, workers, keepalive):
    cache_norm = (cache - 64) / (512 - 64)
    workers_norm = (workers - 2) / (8 - 2)
    keepalive_norm = (keepalive - 10) / (120 - 10)
    rps = 1000 + 800*cache_norm + 600*workers_norm + 300*keepalive_norm
    rps += 400*cache_norm*workers_norm - 200*workers_norm*keepalive_norm
    return rps + np.random.normal(0, 30)

# Матрица планирования
levels = {'-1': 64, '+1': 512}, {'-1': 2, '+1': 8}, {'-1': 10, '+1': 120}
data = []
for a,b,c in product([-1,1], repeat=3):
    data.append({'A': a, 'B': b, 'C': c,
                 'A_val': levels[0][str(a)], 'B_val': levels[1][str(b)], 'C_val': levels[2][str(c)],
                 'RPS': simulate_rps(levels[0][str(a)], levels[1][str(b)], levels[2][str(c)])})
df = pd.DataFrame(data)
df['AB'] = df['A'] * df['B']
df['AC'] = df['A'] * df['C']
df['BC'] = df['B'] * df['C']

# Расчёт эффектов
y = df['RPS']
for factor in ['A', 'B', 'C', 'AB', 'AC', 'BC']:
    effect = y[df[factor]==1].mean() - y[df[factor]==-1].mean()
    print(f"Эффект {factor}: {effect:.1f}")
```

Модель

```
model = ols('RPS ~ A + B + C + AB + AC + BC', data=df).fit()  
print(f"\nR2 = {model.rsquared:.3f}")  
print(model.params)
```

Данная лабораторная работа даёт практические навыки активного планирования эксперимента, которые позволяют минимизировать число тестов и быстро находить оптимальные настройки программных систем.

ЛАБОРАТОРНАЯ РАБОТА №7

Тема: Дробный факторный эксперимент (ДФЭ) и реплики

Время выполнения: 4 академических часа

Цель: освоить методологию дробного факторного эксперимента для сокращения числа опытов при большом количестве факторов, научиться строить ДФЭ $2^{(k-p)}$, анализировать смешивание эффектов (confounding) и выбирать генерирующие соотношения.

1. Актуальность для программной инженерии

При оптимизации сложных программных систем часто приходится учитывать **5, 6 и более факторов**. Полный факторный эксперимент 2^k требует 2^k опытов:

k	2^k	Трудоёмкость
5	32	Приемлемо
6	64	Дорого
7	128	Затруднительно
8	256	Практически нереально

Дробный факторный эксперимент (ДФЭ) позволяет сократить число опытов в 2, 4, 8 и более раз, жертвуя возможностью отдельно оценить некоторые взаимодействия.

ИТ-примеры применения ДФЭ:

Контекст	Число факторов	ДФЭ	Экономия
Настройка параметров веб-сервера (5 параметров)	5	ДФЭ $2^{(5-1)} = 16$ опытов	вместо 32
Оптимизация JVM (7 параметров)	7	ДФЭ $2^{(7-3)} = 16$ опытов	вместо 128

Контекст	Число факторов	ДФЭ	Экономия
Настройка СУБД (6 параметров)	6	ДФЭ $2^{6-2} = 16$ опытов	вместо 64
А/В тестирование множества гипотез	8	ДФЭ $2^{8-4} = 16$ опытов	вместо 256

Основная идея: предполагаем, что взаимодействия высокого порядка (тройные, четверные) пренебрежимо малы, и направляем разрешающую способность на оценку главных эффектов и парных взаимодействий.

2. Теоретическая справка (ДФЭ $2^{(k-p)}$)

Основные понятия:

Термин	Описание
ДФЭ $2^{(k-p)}$	Дробная реплика от полного факторного эксперимента 2^k , содержащая 2^{k-p} опытов
p	Число факторов, эффекты которых смешаны (confounded) с эффектами других факторов
Генерирующее соотношение	Уравнение, задающее новый фактор как произведение существующих
Определяющий контраст	Произведение всех генерирующих соотношений
Смешивание (confounding)	Невозможность раздельно оценить два и более эффекта
Разрешающая способность (resolution)	Характеристика качества ДФЭ (III, IV, V)

Разрешающая способность (по Box-Hunter):

Resolution	Свойства	Пример
III	Главные эффекты смешаны с парными взаимодействиями	ДФЭ $2^{(3-1)}$
IV	Главные эффекты не смешаны с парными; парные смешаны между собой	ДФЭ $2^{(4-1)}$
V	Главные эффекты и парные не смешаны друг с другом	ДФЭ $2^{(5-1)}$

Генерирующие соотношения для $k=5$, $p=1$ (ДФЭ $2^{(5-1)}$):

$$x_5 = x_1 \cdot x_2 \cdot x_3 \cdot x_4$$

Определяющий контраст:

$$I = x_1 x_2 x_3 x_4 x_5$$

Смешивание эффектов:

- Главный эффект x_5 смешан с взаимодействием $x_1 x_2 x_3 x_4$
- x_1 смешан с $x_2 x_3 x_4 x_5$
- $x_1 x_2$ смешан с $x_3 x_4 x_5$

Матрица планирования ДФЭ $2^{(5-1)}$ (16 опытов):

Опыт	x_1	x_2	x_3	x_4	$x_5 = x_1 x_2 x_3 x_4$
1	-1	-1	-1	-1	-1
2	+1	-1	-1	-1	+1
...	...	...	...	...	...
16	+1	+1	+1	+1	+1

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – инженер по производительности. Ваш веб-сервер (Nginx) имеет **5 настраиваемых параметров**, влияющих на пропускную способность (RPS). Проведение полного факторного эксперимента $2^5 = 32$ опыта требует слишком много времени. Вы решаете использовать **ДФЭ $2^{(5-1)} = 16$ опытов**.

Факторы (k=5):

Фактор	Обозначение	Нижний уровень (-1)	Верхний уровень (+1)
Размер кэша (МБ)	A	64	512
Число рабочих процессов	B	2	8
Keepalive timeout (сек)	C	10	120
Буфер отправки (КБ)	D	4	64
Сжатие gzip	E	off (0)	on (1)

Отклик:

rps – количество запросов в секунду.

Задача:

1. Построить матрицу планирования ДФЭ $2^{(5-1)}$ с генерирующим соотношением $E=ABCDE=ABCD$.
2. Провести эксперимент (симуляция) для 16 опытов.
3. Рассчитать главные эффекты и оценить их значимость.
4. Определить, какие эффекты смешаны (confounding).
5. Сравнить результаты ДФЭ с ПФЭ (для оценки потерь).
6. Дать рекомендации по настройке сервера.

4. Исходные данные (генерация откликов)

Создайте функцию, моделирующую RPS, и сгенерируйте данные для ДФЭ.

```
python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```

from scipy import stats
from itertools import product
from statsmodels.formula.api import ols
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)

def simulate_rps(cache_mb, workers, keepalive_sec, buffer_kb, gzip_on):
    """
    Симуляция пропускной способности веб-сервера.
    Реальная модель содержит главные эффекты и несколько парных взаимодействий.
    """
    # Нормировка в [0,1]
    cache_norm = (cache_mb - 64) / (512 - 64)
    workers_norm = (workers - 2) / (8 - 2)
    keepalive_norm = (keepalive_sec - 10) / (120 - 10)
    buffer_norm = (buffer_kb - 4) / (64 - 4)
    gzip_norm = gzip_on # 0 или 1

    # Истинная модель
    base = 800
    effect_cache = 400 * cache_norm
    effect_workers = 500 * workers_norm
    effect_keepalive = 200 * keepalive_norm
    effect_buffer = 150 * buffer_norm
    effect_gzip = -100 * gzip_norm # сжатие замедляет

    # Взаимодействия
    interaction_cache_workers = 300 * cache_norm * workers_norm
    interaction_workers_gzip = -150 * workers_norm * gzip_norm

    rps = (base + effect_cache + effect_workers + effect_keepalive +
           effect_buffer + effect_gzip + interaction_cache_workers +
           interaction_workers_gzip + np.random.normal(0, 25))

    return max(0, int(rps))

# Параметры для ДФЭ 2^(5-1)
# Генерирующее соотношение: E = ABCD
# Полная матрица ПФЭ для первых 4 факторов (2^4 = 16 опытов)

```

```

# Фактор E вычисляется как произведение A*B*C*D

# Натуральные значения уровней
levels = {
    'A': {'-1': 64, '+1': 512},    # cache
    'B': {'-1': 2, '+1': 8},      # workers
    'C': {'-1': 10, '+1': 120},   # keepalive
    'D': {'-1': 4, '+1': 64},     # buffer
    'E': {'-1': 0, '+1': 1}      # gzip (off/on)
}

# Генерация ДФЭ
design_matrix = []
for a, b, c, d in product([-1, 1], repeat=4):
    e = a * b * c * d # генерирующее соотношение

    a_val = levels['A'][str(a)]
    b_val = levels['B'][str(b)]
    c_val = levels['C'][str(c)]
    d_val = levels['D'][str(d)]
    e_val = levels['E'][str(e)]

    rps = simulate_rps(a_val, b_val, c_val, d_val, e_val)

    design_matrix.append({
        'A': a, 'B': b, 'C': c, 'D': d, 'E': e,
        'A_val': a_val, 'B_val': b_val, 'C_val': c_val,
        'D_val': d_val, 'E_val': e_val, 'E_on': 'on' if e_val == 1 else 'off',
        'RPS': rps
    })

df_dfe = pd.DataFrame(design_matrix)
print("=== ДФЭ 2^(5-1) (16 опытов) ===")
print(df_dfe[['A', 'B', 'C', 'D', 'E', 'A_val', 'B_val', 'C_val', 'D_val', 'E_on', 'RPS']].to_string())
print(f"\nСредний RPS: {df_dfe['RPS'].mean():.0f}")

```

5. Ход выполнения (пошаговое руководство)

Шаг 1. Импорт библиотек и подготовка данных

python

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from statsmodels.formula.api import ols
from itertools import product
from scipy.stats import norm

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 5)

print(f"Количество опытов: {len(df_dfe)}")
print(f"Ожидаемое количество:  $2^{(5-1)} = 16$ ")

```

Шаг 2. Расчёт эффектов по ДФЭ

```

python
print("\n" + "="*60)
print("РАСЧЁТ ЭФФЕКТОВ (ДФЭ  $2^{(5-1)}$ )")
print("="*60)

y = df_dfe['RPS'].values
factors = ['A', 'B', 'C', 'D', 'E']

# Добавляем столбцы взаимодействий (которые могут быть оценены)
for combo in [('A', 'B'), ('A', 'C'), ('A', 'D'), ('A', 'E'),
               ('B', 'C'), ('B', 'D'), ('B', 'E'),
               ('C', 'D'), ('C', 'E'), ('D', 'E')]:
    df_dfe[f"{combo[0]}{combo[1]}"] = df_dfe[combo[0]] * df_dfe[combo[1]]

# Расчёт эффектов
effect_values = {}
for col in df_dfe.columns:
    if col in ['A', 'B', 'C', 'D', 'E'] or (len(col) == 2 and col[0] in 'ABCDE' and col[1] in 'ABCDE'):
        if col in df_dfe.columns:
            y_plus = y[df_dfe[col] == 1].mean()
            y_minus = y[df_dfe[col] == -1].mean()
            effect = y_plus - y_minus
            effect_values[col] = effect

# Сортировка по абсолютной величине

```

```

effect_sorted = sorted(effect_values.items(), key=lambda x: abs(x[1]), reverse=True)

print("\nОценённые эффекты (с учётом смешивания):")
for name, eff in effect_sorted[:15]: # mon-15
    print(f" {name:5} : {eff:8.1f} RPS")

```

Шаг 3. Определение смешивания (confounding)

```

python
print("\n" + "="*60)
print("СМЕШИВАНИЕ ЭФФЕКТОВ (CONFOUNDING)")
print("="*60)

# Для ДФЭ 2^(5-1) с генератором E = ABCD
# Определяющий контраст: I = ABCDE

print("Генерирующее соотношение: E = A·B·C·D")
print("Определяющий контраст: I = A·B·C·D·E")
print("\nСмешивание главных эффектов:")
print("  A = B·C·D·E  (A смешан с четырёхфакторным взаимодействием)")
print("  B = A·C·D·E")
print("  C = A·B·D·E")
print("  D = A·B·C·E")
print("  E = A·B·C·D")

print("\nСмешивание парных взаимодействий:")
print("  A·B = C·D·E")
print("  A·C = B·D·E")
print("  A·D = B·C·E")
print("  A·E = B·C·D")
print("  B·C = A·D·E")
print("  B·D = A·C·E")
print("  B·E = A·C·D")
print("  C·D = A·B·E")
print("  C·E = A·B·D")
print("  D·E = A·B·C")

print("\n  Разрешающая способность: V (Resolution V)")
print("  Главные эффекты не смешаны с парными взаимодействиями.")
print("  Парные взаимодействия смешаны с тройными.")

```

Шаг 4. Визуализация эффектов (Pareto chart)

```
python
# Pareto chart
fig, ax = plt.subplots(figsize=(10, 6))
names = [e[0] for e in effect_sorted[:10]]
values = [e[1] for e in effect_sorted[:10]]
colors = ['red' if v < 0 else 'green' for v in values]

bars = ax.barh(names, values, color=colors)
ax.axvline(x=0, color='black', linewidth=0.8)
ax.set_xlabel('Величина эффекта (RPS)')
ax.set_title('Диаграмма Парето эффектов (ДФЭ 2^(5-1))')
ax.grid(axis='x', linestyle='--', alpha=0.7)

for i, (name, val) in enumerate(effect_sorted[:10]):
    ax.text(val + (5 if val > 0 else -35), i, f'{val:.0f}', va='center')

plt.tight_layout()
plt.show()
```

Шаг 5. Статистическая значимость (нормальная вероятностная диаграмма)

```
python
print("\n" + "="*60)
print("НОРМАЛЬНАЯ ВЕРОЯТНОСТНАЯ ДИАГРАММА ЭФФЕКТОВ")
print("="*60)

# Метод: эффекты, распределённые нормально (незначимые), ложатся на прямую
effect_list = [(name, val) for name, val in effect_values.items() if name in ['A', 'B', 'C', 'D', 'E'] or (len(name)==2)]
effect_vals = np.array([e[1] for e in effect_list])
effect_names = [e[0] for e in effect_list]

# Сортировка
sorted_idx = np.argsort(effect_vals)
effect_vals_sorted = effect_vals[sorted_idx]
effect_names_sorted = [effect_names[i] for i in sorted_idx]

# Теоретические квантили нормального распределения
n_effects = len(effect_vals_sorted)
theoretical_quantiles = norm.ppf(np.arange(1, n_effects+1) / (n_effects+1))

# Построение графика
fig, ax = plt.subplots(figsize=(10, 6))
```

```

ax.scatter(theoretical_quantiles, effect_vals_sorted, s=80, alpha=0.7)

# Подписи точек
for i, name in enumerate(effect_names_sorted):
    ax.annotate(name, (theoretical_quantiles[i], effect_vals_sorted[i]),
                fontsize=9, ha='center', va='bottom')

# Линия регрессии для центральных точек (исключая выбросы)
# Для простоты: проводим линию через Q1 и Q3
z1, z3 = np.percentile(theoretical_quantiles, [25, 75])
e1, e3 = np.percentile(effect_vals_sorted, [25, 75])
slope = (e3 - e1) / (z3 - z1)
intercept = e1 - slope * z1
ax.plot([theoretical_quantiles.min(), theoretical_quantiles.max()],
        [intercept + slope * theoretical_quantiles.min(), intercept + slope * theoret
ical_quantiles.max()],
        'r--', lw=2, label='Нормальная линия')

ax.set_xlabel('Теоретические квантили нормального распределения')
ax.set_ylabel('Наблюдаемые эффекты')
ax.set_title('Нормальная вероятностная диаграмма эффектов')
ax.legend()
ax.grid(True, linestyle='--', alpha=0.3)
plt.show()

# Выявление значимых эффектов (отклоняющихся от прямой)
print("\nЭффекты,显著 отклоняющиеся от нормальной линии (потенциально значимые):")
significant_effects = []
for name, val in effect_sorted:
    if abs(val) > 150: # порог, видимый на графике
        significant_effects.append((name, val))
        sign = "+" if val > 0 else "-"
        print(f" {name:5} : {val:8.1f} RPS (возможно значим)")

print("\n Для точной оценки значимости нужны повторные опыты или априорная оценка д
исперсии.")

```

Шаг 6. Построение регрессионной модели (только значимые эффекты)

```

python
print("\n" + "="*60)
print("РЕГРЕССИОННАЯ МОДЕЛЬ (по ДФЭ)")
print("="*60)

```

```

# Выбираем значимые факторы (из Pareto)
significant_factors = ['A', 'B', 'D', 'AB'] # на основе эффектов выше

# Формула
formula = 'RPS ~ ' + ' + '.join(significant_factors)
model_dfe = ols(formula, data=df_dfe).fit()
print(model_dfe.summary())

# Уравнение регрессии в кодированных переменных
print(f"\nМодель RPS = {model_dfe.params['Intercept']:.1f}")
for factor in significant_factors:
    coef = model_dfe.params[factor]
    if coef > 0:
        print(f"          + {coef:.1f}·{factor}")
    else:
        print(f"          - {abs(coef):.1f}·{factor}")

```

Шаг 7. Графический анализ главных эффектов

```

python
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
factors_plot = ['A', 'B', 'C', 'D', 'E']
titles = ['A: Размер кэша', 'B: Число процессов', 'C: Keepalive', 'D: Буфер отправки',
          , 'E: Gzip сжатие']

for i, (factor, title) in enumerate(zip(factors_plot, titles)):
    row, col = i // 3, i % 3
    mean_minus = df_dfe[df_dfe[factor] == -1]['RPS'].mean()
    mean_plus = df_dfe[df_dfe[factor] == 1]['RPS'].mean()

    axes[row, col].bar(['-1 (низкий)', '+1 (высокий)'], [mean_minus, mean_plus],
                       color=['steelblue', 'coral'])
    axes[row, col].set_ylabel('Средний RPS')
    axes[row, col].set_title(title)
    axes[row, col].grid(axis='y', linestyle='--', alpha=0.7)

    effect = mean_plus - mean_minus
    axes[row, col].text(0.5, min(mean_minus, mean_plus) - 25, f'Эффект = {effect:.0f}',
                        ha='center', fontsize=10)

# Убираем пустой подграфик

```

```
axes[1, 2].axis('off')
plt.tight_layout()
plt.show()
```

Шаг 8. Сравнение ДФЭ с ПФЭ (анализ потерь)

```
python
print("\n" + "="*60)
print("СРАВНЕНИЕ ДФЭ С ПОЛНЫМ ФАКТОРНЫМ ЭКСПЕРИМЕНТОМ")
print("="*60)

print("Преимущества ДФЭ:")
print(f"    • Число опытов: {len(df_dfe)} вместо 32 (экономия 50%)")
print(f"    • Экономия времени: ~{(1 - len(df_dfe)/32)*100:.0f}%")
print(f"    • Оценены главные эффекты и некоторые парные взаимодействия")

print("\nНедостатки и потери ДФЭ:")
print("    • Невозможно разделить оценить: AB и CDE")
print("    • AC и BDE, AD и BCE, AE и BCD")
print("    • BC и ADE, BD и ACE, BE и ACD")
print("    • CD и ABE, CE и ABD, DE и ABC")

print("\nВывод: При предположении, что тройные и четверные взаимодействия")
print("пренебрежимо малы, ДФЭ 25-1 даёт корректные оценки главных")
print("эффектов и парных взаимодействий (Resolution V).")
```

Шаг 9. Рекомендации по оптимизации

```
python
print("\n" + "="*60)
print("РЕКОМЕНДАЦИИ ПО НАСТРОЙКЕ ВЕБ-СЕРВЕРА")
print("="*60)

# Лучшая конфигурация из проведённых опытов
best_row = df_dfe.loc[df_dfe['RPS'].idxmax()]
print(f"Лучшая конфигурация из {len(df_dfe)} опытов ДФЭ:")
print(f"    A (cache)      = {best_row['A_val']} МБ")
print(f"    B (workers)    = {best_row['B_val']}")
print(f"    C (keepalive)  = {best_row['C_val']} сек")
print(f"    D (buffer)     = {best_row['D_val']} КБ")
print(f"    E (gzip)       = {best_row['E_on']}")
print(f"    RPS = {best_row['RPS']:.0f}")
```

```

# Направления изменения для значимых факторов
print("\nНаправления для увеличения RPS:")
for factor in significant_factors:
    if factor == 'AB':
        continue
    effect = effect_values.get(factor, 0)
    if effect > 100:
        print(f" • {factor}: увеличить (эффект +{effect:.0f} RPS)")
    elif effect < -100:
        print(f" • {factor}: уменьшить (эффект {effect:.0f} RPS)")

# Рекомендация по следующему шагу
print("\nРекомендация для следующего эксперимента:")
print(" 1. Зафиксировать C и E на уровнях, дающих лучший результат.")
print(" 2. Провести ПФЭ 2^3 для факторов A, B, D (8 опытов).")
print(" 3. Построить поверхность отклика для точной оптимизации.")

```

Шаг 10. Проверка адекватности модели

```

python
# Предсказанные значения
df_dfe['RPS_pred'] = model_dfe.predict(df_dfe)
df_dfe['residual'] = df_dfe['RPS'] - df_dfe['RPS_pred']

# График фактических vs предсказанных
plt.figure(figsize=(8, 6))
plt.scatter(df_dfe['RPS'], df_dfe['RPS_pred'], s=100, alpha=0.7)
plt.plot([df_dfe['RPS'].min(), df_dfe['RPS'].max()],
         [df_dfe['RPS'].min(), df_dfe['RPS'].max()], 'r--', lw=2)
plt.xlabel('Фактический RPS')
plt.ylabel('Предсказанный RPS')
plt.title(f'Фактические vs Предсказанные ( $R^2 = \{model\_dfe.rsquared:.3f\}$ )')
plt.grid(True, linestyle='--', alpha=0.3)
plt.show()

print(f"\nR2 модели: {model_dfe.rsquared:.3f}")
print(f"Скорректированный R2: {model_dfe.rsquared_adj:.3f}")
print(f"RMSE: {np.sqrt(model_dfe.mse_resid):.1f} RPS")

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов**. Необходимо выполнить полный анализ по шагам 1–10.

Вар.	k	ДФЭ	Факторы	Контекст
1	5	$2^{(5-1)}$	A,B,C,D,E	Веб-сервер (как в примере)
2	6	$2^{(6-2)}$	A,B,C,D,E,F	JVM параметры (Xms,Xmx,GC, etc.)
3	7	$2^{(7-3)}$	A...G	Настройка PostgreSQL
4	5	$2^{(5-1)}$	A...E	Алгоритм сжатия данных
5	6	$2^{(6-2)}$	A...F	Настройка CDN
6	4	$2^{(4-1)}$	A,B,C,D	Балансировщик нагрузки

Указания для варианта 2 (JVM, k=6, ДФЭ $2^{(6-2)} = 16$ опытов):

```
python
# Генерирующие соотношения для  $2^{(6-2)}$ 
# E = ABC, F = BCD
# Определяющий контраст: I = ABCE = BCDF = ADEF
# Разрешающая способность: IV
```

7. Структура отчета

1. **Цель работы**
2. **Описание факторов и отклика** (k=5, уровни)
3. **Матрица планирования ДФЭ** (16 опытов, кодированные значения)
4. **Генерирующие соотношения и смешивание**
5. **Расчёт эффектов** (таблица, Pareto chart)
6. **Нормальная вероятностная диаграмма** (выявление значимых)
7. **Регрессионная модель** (коэффициенты, R^2)
8. **Сравнение с ПФЭ** (экономия, потери)
9. **Рекомендации по оптимизации**
10. **Ответы на контрольные вопросы**

8. Контрольные вопросы (для защиты)

1. Что такое дробный факторный эксперимент и зачем он нужен?
2. Что означает обозначение ДФЭ $2^{(5-1)}$? Сколько опытов в таком плане?

3. Что такое генерирующее соотношение? Приведите пример.
4. Что такое определяющий контраст и как он используется?
5. Что означает «смешивание (confounding) эффектов»?
6. Что такое разрешающая способность (resolution) и какие значения бывают?
7. Чем Resolution V отличается от Resolution III?
8. Как построить матрицу ДФЭ $2^{(5-1)}$ на основе ПФЭ 2^4 ?
9. Какой главный недостаток ДФЭ по сравнению с ПФЭ?
10. Когда применение ДФЭ оправдано, а когда нет?

9. Полный код в одном блоке (для копирования)

```
python
# Полный код ДФЭ  $2^{(5-1)}$ 
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from itertools import product
from statsmodels.formula.api import ols
from scipy.stats import norm

np.random.seed(42)

# Симуляция отклика
def simulate(a_val, b_val, c_val, d_val, e_val):
    a_n = (a_val - 64) / (512 - 64)
    b_n = (b_val - 2) / (8 - 2)
    c_n = (c_val - 10) / (120 - 10)
    d_n = (d_val - 4) / (64 - 4)
    rps = 800 + 400*a_n + 500*b_n + 200*c_n + 150*d_n - 100*e_val
    rps += 300*a_n*b_n - 150*b_n*e_val
    return rps + np.random.normal(0, 25)

# ДФЭ  $2^{(5-1)}$ :  $E = ABCD$ 
levels = {'A': {'-1': 64, '+1': 512}, 'B': {'-1': 2, '+1': 8}, 'C': {'-1': 10, '+1': 120},
          'D': {'-1': 4, '+1': 64}, 'E': {'-1': 0, '+1': 1}}
data = []
for a,b,c,d in product([-1,1], repeat=4):
    e = a*b*c*d
    rps = simulate(levels['A'][str(a)], levels['B'][str(b)], levels['C'][str(c)],
                  levels['D'][str(d)], levels['E'][str(e)])
    data.append({'A':a, 'B':b, 'C':c, 'D':d, 'E':e, 'RPS':rps})
df = pd.DataFrame(data)
```

Расчёт эффектов

```
for f in ['A', 'B', 'C', 'D', 'E']:
    eff = df[df[f]==1]['RPS'].mean() - df[df[f]==-1]['RPS'].mean()
    print(f"Эффект {f}: {eff:.1f}")
```

Модель

```
model = ols('RPS ~ A + B + D', data=df).fit()
print(f"\nR² = {model.rsquared:.3f}")
print(model.params)
```

Данная лабораторная работа даёт практические навыки сокращения объёма экспериментальных исследований, что критически важно при оптимизации сложных программных систем с большим числом параметров.

ЛАБОРАТОРНАЯ РАБОТА №8

Тема: Анализ временных рядов в А/Б тестировании

Время выполнения: 4 академических часа

Цель: освоить методы анализа временных рядов для оценки эффекта вмешательства (релиза, фичи, изменения конфигурации) в условиях сезонности и трендов, научиться применять interrupted time series analysis (ITS) и CUSUM-диаграммы.

1. Актуальность для программной инженерии

Классическое А/Б тестирование предполагает одновременное сравнение двух групп (контрольной и экспериментальной) в один и тот же период времени. Однако во многих случаях **параллельное тестирование невозможно**:

- Релиз новой фичи выкатывается на всех пользователей одновременно.
- Изменение конфигурации сервера нельзя применить только к части трафика.
- Оценивается долгосрочный эффект (например, удержание пользователей).

В таких случаях используется анализ временных рядов с точкой вмешательства (Interrupted Time Series Analysis – ITS).

ИТ-примеры:

Сценарий	Метрика	Вмешательство
Релиз нового алгоритма рекомендаций	Время на сайте (мин)	Дата релиза
Оптимизация кэширования	Частота ошибок 5xx	Дата деплоя
Изменение цены подписки	Конверсия в платную	Дата изменения
Перенос на новую инфраструктуру	Время отклика API	Дата миграции
Внедрение автоматического тестирования	Частота багов в проде	Дата внедрения

Основная задача: отделить эффект вмешательства от естественных трендов и сезонности.

2. Теоретическая справка (Interrupted Time Series)

Модель ITS (сегментированная регрессия):

$$Y_t = \beta_0 + \beta_1 T_t + \beta_2 X_t + \beta_3 (T_t - T_0) X_t + \varepsilon_t$$

Где:

- Y_t – значение метрики в момент времени t
- T_t – время (например, дни с начала наблюдения)
- X_t – индикатор вмешательства (0 до, 1 после)
- T_0 – момент вмешательства

Интерпретация коэффициентов:

Коэффициент	Обозначение	Интерпретация
β_0	Уровень до	Базовое значение в момент 0
β_1	Тренд до	Ежемесячное/ежедневное изменение до вмешательства
β_2	Мгновенное изменение	Скачок сразу после вмешательства
β_3	Изменение тренда	Разница в наклоне после вмешательства

CUSUM (кумулятивная сумма):

$$S_t = \sum_{i=1}^t (Y_i - \bar{Y})$$

CUSUM позволяет визуально обнаружить момент устойчивого изменения среднего значения.

Проверка предпосылок ITS:

- Отсутствие автокорреляции остатков (тест Дарбина-Уотсона)
- Стационарность (тест Дики-Фуллера для остатков)

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – SRE-инженер. Ваша команда выкатила оптимизацию кэширования на всех серверах **15 марта 2025 года**. У вас есть ежедневные данные о частоте ошибок HTTP 5xx (в процентах от всех запросов) за 60 дней до релиза и 40 дней после.

Данные:

- Период: 60 дней до вмешательства (15.01.2025 – 14.03.2025) + 40 дней после (15.03.2025 – 23.04.2025)
- Частота ошибок 5xx (%) – метрика, которую нужно улучшить (снизить)
- Естественный тренд: постепенный рост ошибок до вмешательства (деградация)

Задача:

1. Визуализировать временной ряд с отметкой точки вмешательства.
2. Построить сегментированную регрессию (ITS) для оценки эффекта.
3. Проверить предпосылки (автокорреляция, стационарность).
4. Рассчитать CUSUM для обнаружения момента изменения.
5. Интерпретировать результаты: было ли улучшение? насколько значимое?
6. Сделать вывод о эффективности оптимизации.

4. Исходные данные (генерация)

Создайте временной ряд с трендом, сезонностью и эффектом вмешательства.

```
python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.stattools import durbin_watson
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
```

```

# Параметры
n_before = 60      # дней до релиза
n_after = 40       # дней после релиза
intervention_day = n_before # индекс точки вмешательства (60, если с 0)

# Временная шкала
days = np.arange(n_before + n_after)
dates = pd.date_range(start='2025-01-15', periods=n_before + n_after, freq='D')

# Базовые компоненты
trend_before = np.linspace(1.2, 1.8, n_before) # растущий тренд (деградация)
trend_after = np.linspace(1.6, 1.3, n_after)    # нисходящий тренд после оптимизации

# Сезонность (недельная, 7 дней)
seasonality = 0.2 * np.sin(2 * np.pi * days / 7)

# Шум
noise = np.random.normal(0, 0.1, n_before + n_after)

# Эффект вмешательства (мгновенное снижение на 0.4%)
intervention_effect = np.zeros(n_before + n_after)
intervention_effect[intervention_day:] = -0.4

# Формирование ряда
error_rate = np.concatenate([trend_before, trend_after]) + seasonality + noise + intervention_effect
error_rate = np.maximum(error_rate, 0.1) # неотрицательные значения

# Создание DataFrame
df = pd.DataFrame({
    'date': dates,
    'day': days,
    'error_rate_pct': error_rate,
    'after_intervention': [0] * n_before + [1] * n_after
})

# Добавим время (для регрессии)
df['time'] = df['day']
df['time_after'] = (df['day'] - intervention_day) * df['after_intervention']
df['time_after'] = np.maximum(df['time_after'], 0) # обнуляем до вмешательства

print("Первые 10 строк:")

```

```

print(df.head(10))
print("\nПоследние 10 строк:")
print(df.tail(10))
print(f"\nСтатистика до вмешательства (первые {n_before} дней):")
print(df[df['after_intervention'] == 0]['error_rate_pct'].describe())
print(f"\nСтатистика после вмешательства (последние {n_after} дней):")
print(df[df['after_intervention'] == 1]['error_rate_pct'].describe())

```

5. Ход выполнения (пошаговое руководство)

Шаг 1. Импорт библиотек и визуализация ряда

```

python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.stattools import durbin_watson
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (14, 6)

# Визуализация временного ряда
fig, ax = plt.subplots(figsize=(14, 6))
ax.plot(df['date'], df['error_rate_pct'], 'b-', linewidth=1.5, label='Частота ошибок 5xx')
ax.axvline(x=df['date'].iloc[intervention_day], color='red', linestyle='--',
           linewidth=2, label='Релиз оптимизации (15.03.2025)')
ax.set_xlabel('Дата')
ax.set_ylabel('Ошибки 5xx (%)')
ax.set_title('Временной ряд: частота ошибок HTTP 5xx до и после оптимизации кэширования')
ax.legend()
ax.grid(True, linestyle='--', alpha=0.3)

# Добавим средние линии
mean_before = df[df['after_intervention'] == 0]['error_rate_pct'].mean()

```

```

mean_after = df[df['after_intervention'] == 1]['error_rate_pct'].mean()
ax.axhline(y=mean_before, color='blue', linestyle=':', alpha=0.7, label=f'Среднее до: {mean_before:.2f}%')
ax.axhline(y=mean_after, color='green', linestyle=':', alpha=0.7, label=f'Среднее после: {mean_after:.2f}%')
ax.legend()

plt.tight_layout()
plt.show()

```

Шаг 2. Сегментированная регрессия (ITS)

```

python
print("\n" + "="*60)
print("SEGMENTED REGRESSION (INTERRUPTED TIME SERIES)")
print("="*60)

# Модель:  $Y = \theta_0 + \theta_1 \cdot time + \theta_2 \cdot after + \theta_3 \cdot time\_after$ 
#  $time\_after = (day - intervention\_day) \cdot after$ 
X = df[['time', 'after_intervention', 'time_after']]
X = sm.add_constant(X)
y = df['error_rate_pct']

model_its = sm.OLS(y, X).fit()
print(model_its.summary())

# Извлечение коэффициентов
beta0 = model_its.params['const']
beta1 = model_its.params['time']
beta2 = model_its.params['after_intervention']
beta3 = model_its.params['time_after']

print("\n=== ИНТЕРПРЕТАЦИЯ КОЭФФИЦИЕНТОВ ===")
print(f"β0 (уровень в начале) = {beta0:.4f}%")
print(f"β1 (тренд до) = {beta1:.4f}%/день → {'рост' if beta1 > 0 else 'снижение'} {abs(beta1):.4f}%/день")
print(f"β2 (мгновенное изменение) = {beta2:.4f}% → {'ухудшение' if beta2 > 0 else 'улучшение'} на {abs(beta2):.4f}%")
print(f"β3 (изменение тренда) = {beta3:.4f}%/день → {'ускорение роста' if beta3 > 0 else 'замедление/снижение'} {abs(beta3):.4f}%/день")

# Общий эффект через 30 дней после релиза
effect_30_days = beta2 + beta3 * 30

```

```
print(f"\nСовокупный эффект через 30 дней: {effect_30_days:.4f}%")
```

Шаг 3. Визуализация модели ITS

```
python
```

```
# Предсказанные значения
```

```
df['predicted'] = model_its.predict(X)
```

```
# Прогноз "что было бы без вмешательства" (контрфактуал)
```

```
df['counterfactual'] = beta0 + beta1 * df['time']
```

```
fig, ax = plt.subplots(figsize=(14, 6))
```

```
ax.plot(df['date'], df['error_rate_pct'], 'b-', linewidth=1.5, alpha=0.6, label='Фактические данные')
```

```
ax.plot(df['date'], df['predicted'], 'r-', linewidth=2, label='ITS модель (сегментированная регрессия)')
```

```
ax.plot(df['date'], df['counterfactual'], 'g--', linewidth=2, label='Контрфактуал (без вмешательства)', alpha=0.8)
```

```
ax.axvline(x=df['date'].iloc[intervention_day], color='red', linestyle='--', linewidth=2, label='Момент вмешательства')
```

```
ax.set_xlabel('Дата')
```

```
ax.set_ylabel('Ошибки 5xx (%)')
```

```
ax.set_title('Interrupted Time Series Analysis: фактический vs контрфактуальный тренд')
```

```
ax.legend()
```

```
ax.grid(True, linestyle='--', alpha=0.3)
```

```
# Закрашиваем область эффекта
```

```
ax.fill_between(df['date'][intervention_day:],  
                df['counterfactual'][intervention_day:],  
                df['predicted'][intervention_day:],  
                alpha=0.3, color='green', label='Эффект вмешательства')
```

```
plt.tight_layout()
```

```
plt.show()
```

Шаг 4. Проверка предпосылок (автокорреляция)

```
python
```

```
print("\n" + "="*60)
```

```
print("ПРОВЕРКА ПРЕДПОСЫЛОК МОДЕЛИ")
```

```
print("="*60)
```

```

# Остатки
residuals = model_its.resid

# 1. Тест Дарбина-Уотсона (автокорреляция)
dw_stat = durbin_watson(residuals)
print(f"Тест Дарбина-Уотсона: DW = {dw_stat:.3f}")
if dw_stat < 1.5:
    print(" ☐ Положительная автокорреляция (DW < 1.5)")
elif dw_stat > 2.5:
    print(" ☐ Отрицательная автокорреляция (DW > 2.5)")
else:
    print(" ☒ Автокорреляция отсутствует")

# 2. ACF (автокорреляционная функция)
fig, axes = plt.subplots(1, 2, figsize=(14, 4))
plot_acf(residuals, lags=20, ax=axes[0])
axes[0].set_title('Автокорреляционная функция (ACF) остатков')

# 3. Q-Q plot остатков
stats.probplot(residuals, dist="norm", plot=axes[1])
axes[1].set_title('Q-Q plot остатков')

plt.tight_layout()
plt.show()

# 4. Тест на нормальность
shapiro_stat, shapiro_p = stats.shapiro(residuals)
print(f"\nТест Шапиро-Уилка (нормальность): p = {shapiro_p:.4f}")
if shapiro_p > 0.05:
    print(" ☒ Остатки нормально распределены")
else:
    print(" ☐ Остатки не нормальны (но для ITS это не критично при большом n)")

# 5. Тест Дики-Фуллера (стационарность остатков)
adf_stat, adf_p, _, _, _, _ = adfuller(residuals)
print(f"\nТест Дики-Фуллера (стационарность остатков): p = {adf_p:.4f}")
if adf_p < 0.05:
    print(" ☒ Остатки стационарны")
else:
    print(" ☐ Остатки не стационарны (возможен тренд в остатках)")

```

Шаг 5. Анализ CUSUM (кумулятивная сумма)

```

python
print("\n" + "="*60)
print("CUSUM АНАЛИЗ (ОБНАРУЖЕНИЕ МОМЕНТА ИЗМЕНЕНИЯ)")
print("="*60)

# Расчёт CUSUM
mean_overall = df['error_rate_pct'].mean()
df['cusum'] = (df['error_rate_pct'] - mean_overall).cumsum()

# CUSUM для остатков
df['cusum_residual'] = residuals.cumsum()

# Визуализация CUSUM
fig, axes = plt.subplots(2, 1, figsize=(14, 8))

# CUSUM исходного ряда
axes[0].plot(df['date'], df['cusum'], 'b-', linewidth=1.5)
axes[0].axvline(x=df['date'].iloc[intervention_day], color='red', linestyle='--', linewidth=2)
axes[0].axhline(y=0, color='black', linestyle='-', linewidth=0.5)
axes[0].set_xlabel('Дата')
axes[0].set_ylabel('CUSUM')
axes[0].set_title('CUSUM диаграмма: обнаружение момента изменения среднего')
axes[0].grid(True, linestyle='--', alpha=0.3)

# Добавим доверительные границы (приблизительные)
std_dev = df['error_rate_pct'].std()
threshold = 2 * std_dev * np.sqrt(len(df))
axes[0].fill_between(df['date'], -threshold, threshold, alpha=0.2, color='gray', label='Доверительная область')
axes[0].legend()

# CUSUM остатков
axes[1].plot(df['date'], df['cusum_residual'], 'g-', linewidth=1.5)
axes[1].axvline(x=df['date'].iloc[intervention_day], color='red', linestyle='--', linewidth=2)
axes[1].axhline(y=0, color='black', linestyle='-', linewidth=0.5)
axes[1].set_xlabel('Дата')
axes[1].set_ylabel('CUSUM остатков')
axes[1].set_title('CUSUM остатков: проверка устойчивости модели')
axes[1].grid(True, linestyle='--', alpha=0.3)

```

```

plt.tight_layout()
plt.show()

# Интерпретация CUSUM
print("Интерпретация CUSUM диаграммы:")
if abs(df['cusum'].iloc[-1]) > threshold:
    print("    • CUSUM пересек доверительные границы → обнаружено значимое изменение")
else:
    print("    • CUSUM в пределах доверительной области → изменение не обнаружено")

```

Шаг 6. Проверка значимости эффекта с помощью бутстрапа

```

python
print("\n" + "="*60)
print("БУТСТРАП-ОЦЕНКА ЭФФЕКТА ВМЕШАТЕЛЬСТВА")
print("="*60)

# Бутстреп для доверительных интервалов эффекта
n_bootstrap = 1000
bootstrap_effects = []

np.random.seed(42)
for _ in range(n_bootstrap):
    # Ресэмплинг остатков
    boot_residuals = np.random.choice(residuals, size=len(residuals), replace=True)
    boot_y = df['predicted'] + boot_residuals

    # Переоценка модели
    boot_model = sm.OLS(boot_y, X).fit()
    boot_beta2 = boot_model.params['after_intervention']
    boot_beta3 = boot_model.params['time_after']
    boot_effect_30 = boot_beta2 + boot_beta3 * 30
    bootstrap_effects.append(boot_effect_30)

bootstrap_effects = np.array(bootstrap_effects)
ci_lower = np.percentile(bootstrap_effects, 2.5)
ci_upper = np.percentile(bootstrap_effects, 97.5)

print(f"Мгновенное изменение ( $\beta_2$ ): {beta2:.4f}%")
print(f"95% ДИ (бутстреп): [{ci_lower:.4f}%, {ci_upper:.4f}%]")

if ci_lower > 0:
    print("    ✓Эффект значимо положительный (ухудшение)")

```

```

elif ci_upper < 0:
    print("    ✓ Эффект значимо отрицательный (улучшение)")
else:
    print("    □ Эффект статистически не значим (ДИ пересекает 0)")

```

Шаг 7. Декомпозиция временного ряда (тренд, сезонность)

python

```

from statsmodels.tsa.seasonal import seasonal_decompose

# Декомпозиция (требуется установленная частота)
df_ts = df.set_index('date')['error_rate_pct']
df_ts = df_ts.asfreq('D')

# Линейная интерполяция для пропусков (если есть)
if df_ts.isnull().any():
    df_ts = df_ts.interpolate()

# Декомпозиция
decomposition = seasonal_decompose(df_ts, model='additive', period=7)

fig, axes = plt.subplots(4, 1, figsize=(14, 10))
decomposition.observed.plot(ax=axes[0], legend=False)
axes[0].set_title('Исходный ряд')
axes[0].axvline(x=df['date'].iloc[intervention_day], color='red', linestyle='--')

decomposition.trend.plot(ax=axes[1], legend=False)
axes[1].set_title('Тренд')
axes[1].axvline(x=df['date'].iloc[intervention_day], color='red', linestyle='--')

decomposition.seasonal.plot(ax=axes[2], legend=False)
axes[2].set_title('Сезонность (недельная)')

decomposition.resid.plot(ax=axes[3], legend=False)
axes[3].set_title('Остатки (случайная компонента)')
axes[3].axvline(x=df['date'].iloc[intervention_day], color='red', linestyle='--')

plt.tight_layout()
plt.show()

```

Шаг 8. Сравнительный анализ: до vs после (простой t-test)

python

```

print("\n" + "="*60)
print("СРАВНЕНИЕ ДО vs ПОСЛЕ (ПРОСТОЙ T-TEST)")
print("="*60)

before = df[df['after_intervention'] == 0]['error_rate_pct']
after = df[df['after_intervention'] == 1]['error_rate_pct']

t_stat, t_p = stats.ttest_ind(before, after, equal_var=False)
print(f"Среднее до: {before.mean():.4f}%")
print(f"Среднее после: {after.mean():.4f}%")
print(f"Абсолютное изменение: {after.mean() - before.mean():.4f}%")
print(f"Относительное изменение: {(after.mean() - before.mean()) / before.mean() * 100:.2f}%")
print(f"t-test: p = {t_p:.6f}")

if t_p < 0.05:
    print("    ✓Различия статистически значимы")
else:
    print("    ✗Различия не значимы")

print("\n⚠ Предупреждение: простой t-test игнорирует временную структуру данных")
print("    и может давать ошибочные результаты при наличии тренда или автокорреляции.")
)
print("    Доверять следует ITS-модели (сегментированной регрессии).")

```

Шаг 9. Итоговый вывод

```

python
print("\n" + "="*60)
print("ИТОГОВЫЙ ВЫВОД")
print("="*60)

print("ОЦЕНКА ЭФФЕКТИВНОСТИ ОПТИМИЗАЦИИ КЭШИРОВАНИЯ:")
print("-" * 50)

if beta2 < 0 and model_its.pvalues['after_intervention'] < 0.05:
    print(f"    ✓Мгновенное улучшение: частота ошибок снизилась на {abs(beta2):.3f}% сразу после релиза.")
elif beta2 > 0 and model_its.pvalues['after_intervention'] < 0.05:
    print(f"    ✗Мгновенное ухудшение: частота ошибок выросла на {beta2:.3f}% сразу после релиза.")
else:
    print("    Мгновенное изменение статистически не значимо.")

```

```

if beta3 < 0 and model_its.pvalues['time_after'] < 0.05:
    print(f"✅Изменение тренда: после релиза ошибки снижаются на {abs(beta3):.3f}%/день дополнительно.")
elif beta3 > 0 and model_its.pvalues['time_after'] < 0.05:
    print(f"❌Изменение тренда: после релиза ошибки растут на {beta3:.3f}%/день быстрее.")
else:
    print("❏ Изменение тренда статистически не значимо.")

print(f"\nСовокупный эффект через 30 дней: {effect_30_days:.3f}%")
if effect_30_days < 0:
    print(f" → Через месяц после релиза ошибки стали на {abs(effect_30_days):.2f}% ниже ожидаемых.")
else:
    print(f" → Через месяц после релиза ошибки стали на {effect_30_days:.2f}% выше ожидаемых.")

print("\nРЕКОМЕНДАЦИИ:")
if beta2 < 0 or beta3 < 0:
    print(" • Оптимизация кэширования показала положительный эффект.")
    print(" • Рекомендуется зафиксировать изменения и продолжить мониторинг.")
    print(" • Рассмотреть возможность дальнейшей оптимизации.")
else:
    print(" • Оптимизация не дала ожидаемого эффекта или привела к ухудшению.")
    print(" • Рекомендуется откатить изменение и провести дополнительный анализ.")

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов**. Необходимо выполнить полный анализ по шагам 1–9.

Вар.	Метрика	Вмешательство	Характер эффекта
1	Ошибки 5xx (%)	Оптимизация кэша	Снижение (как в примере)
2	Время ответа API (мс)	Новый балансировщик	Снижение задержки
3	Конверсия в покупку (%)	Новый дизайн кнопки	Рост конверсии

Вар.	Метрика	Вмешательство	Характер эффекта
4	Время сборки CI (мин)	Параллелизация тестов	Снижение времени
5	Потребление памяти (МБ)	Новый алгоритм	Снижение памяти
6	RPS (запросов/сек)	Увеличение кэша	Рост пропускной способности

Указание по генерации для варианта 2 (пример):

```
python
# Время ответа API: тренд вверх (деградация) + снижение после вмешательства
trend_before = np.linspace(120, 150, n_before)
trend_after = np.linspace(148, 110, n_after)
intervention_effect = np.zeros(n_before + n_after)
intervention_effect[intervention_day:] = -20 # мгновенное снижение на 20 мс
```

7. Структура отчета

1. **Цель работы**
2. **Описание данных** (метрика, период, момент вмешательства)
3. **Визуализация временного ряда**
4. **Модель ITS** (коэффициенты, интерпретация)
5. **Проверка предпосылок** (автокорреляция, стационарность)
6. **CUSUM анализ** (обнаружение момента изменения)
7. **Сравнение с t-test** (почему ITS лучше)
8. **Вывод о эффективности вмешательства**
9. **Ответы на контрольные вопросы**

8. Контрольные вопросы (для защиты)

1. Чем ITS отличается от классического А/Б тестирования?
2. Что такое контрфактуал в контексте ITS?
3. Как интерпретировать коэффициенты β_2 и β_3 в сегментированной регрессии?
4. Что такое CUSUM и как он помогает обнаружить момент изменения?
5. Почему простой t-test (до vs после) может давать неверные результаты?
6. Как проверить наличие автокорреляции в остатках?
7. Что означает тест Дарбина-Уотсона и какие значения считаются хорошими?
8. Зачем нужна декомпозиция временного ряда (тренд, сезонность)?

9. В каких случаях ITS неприменим?
10. Как оценить долгосрочный эффект вмешательства (через 30/60 дней)?

9. Полный код в одном блоке (для копирования)

```
python
# Полный код ITS анализа

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy import stats
from statsmodels.stats.stattools import durbin_watson

# Генерация данных
np.random.seed(42)
n_before, n_after = 60, 40
trend = np.concatenate([np.linspace(1.2, 1.8, n_before), np.linspace(1.6, 1.3, n_after)])
season = 0.2 * np.sin(2 * np.pi * np.arange(n_before + n_after) / 7)
effect = np.concatenate([np.zeros(n_before), -0.4 * np.ones(n_after)])
y = trend + season + effect + np.random.normal(0, 0.1, n_before + n_after)

# Подготовка данных
df = pd.DataFrame({'y': y, 'time': np.arange(len(y)), 'after': [0]*n_before + [1]*n_after})
df['time_after'] = df['time'] * df['after']
df['time_after'] -= n_before * df['after']

# ITS модель
X = sm.add_constant(df[['time', 'after', 'time_after']])
model = sm.OLS(df['y'], X).fit()
print(model.summary())

# CUSUM
df['cusum'] = (df['y'] - df['y'].mean()).cumsum()
plt.plot(df['cusum'])
plt.axvline(x=n_before, color='red')
plt.title('CUSUM')
plt.show()
```

Данная лабораторная работа даёт практические навыки оценки эффекта изменений в программных системах в условиях, когда классическое А/Б тестирование невозможно, что критически важно для SRE, DevOps и инженеров по производительности.

ЛАБОРАТОРНАЯ РАБОТА №9

Тема: Робастное проектирование (метод Тагучи)

Время выполнения: 4 академических часа

Цель: освоить методологию робастного проектирования для создания систем, устойчивых к воздействию неуправляемых (шумовых) факторов, научиться строить ортогональные массивы, рассчитывать отношения сигнал/шум (SNR) и оптимизировать параметры.

1. Актуальность для программной инженерии

В реальных условиях программные системы работают в **изменчивой среде**: меняется нагрузка, конфигурация железа, версии библиотек, сетевые задержки. Хорошая система должна не только показывать высокую производительность в идеальных условиях, но и **сохранять стабильность** при воздействии шумов.

Метод Тагучи – это подход к проектированию, который минимизирует чувствительность отклика к неуправляемым факторам.

ИТ-примеры применения метода Тагучи:

Система	Управляемые факторы (внутренние)	Шумовые факторы (внешние)	Отклик (цель)
JVM настройка	Xms, Xmx, GC тип, размер поколений	RPS, размер объектов, время суток	Пауза GC (минимизировать)
Балансировщик	Алгоритм, таймауты, буфера	Количество пользователей, тип запросов	Время отклика (стабильность)
Система кэширования	Размер кэша, политика вытеснения	Паттерн доступа, коэффициент попаданий	Hit rate (максимизировать)
Алгоритм сжатия	Уровень сжатия, размер блока	Тип данных (текст, бинарные), энтропия	Размер результата + время

Система	Управляемые факторы (внутренние)	Шумовые факторы (внешние)	Отклик (цель)
Автомасштабирование	Пороги срабатывания, шаг масштабирования	Изменение нагрузки, задержки старта	Время реакции (минимальное)

Три этапа робастного проектирования по Тагучи:

1. **Системное проектирование** – выбор архитектуры и принципов работы.
2. **Параметрическое проектирование** – оптимизация параметров для минимизации чувствительности к шумам.
3. **Допусковое проектирование** – определение допустимых отклонений параметров.

2. Теоретическая справка (метод Тагучи)

Ключевые понятия:

Термин	Обозначение	Описание
Управляемые факторы (Control factors)	A, B, C, ...	Параметры, которые мы можем задать (настройки системы)
Шумовые факторы (Noise factors)	N1, N2, ...	Неуправляемые факторы, вызывающие вариацию отклика
Ортогональный массив	OA(N, k, s, t)	Таблица планирования эксперимента (например, L9, L18, L27)
Внутренний массив (Inner array)	–	План для управляемых факторов
Внешний массив (Outer array)	–	План для шумовых факторов (часто L4 или L9)
Функция потерь Тагучи	$L(y) = k \cdot (y - m)^2$	Квадратичные потери при отклонении от цели

Термин	Обозначение	Описание
Отношение сигнал/шум (SNR)	$\eta = -10 \cdot \log(\text{MSD})$	Метрика робастности (чем выше, тем лучше)

Типы задач и соответствующие SNR:

Тип задачи	Цель	Формула MSD	SNR (η)
Номинальное – лучше (Nominal-the-best)	$y = m$ (целевое значение)	$\frac{1}{n} \sum (y_i - m)^2$	$10 \log \left(\frac{y^2 s^2}{s_y^2} \right) = 10 \log \left(\frac{s_y^2}{y^2} \right)$
Меньше – лучше (Smaller-the-better)	Минимизировать y	$\frac{1}{n} \sum y_i^2$	$-10 \log \left(\frac{1}{n} \sum y_i^2 \right) = -10 \log \left(\frac{1}{n} \sum y_i^2 \right)$
Больше – лучше (Larger-the-better)	Максимизировать y	$\frac{1}{n} \sum \frac{1}{y_i^2}$	$-10 \log \left(\frac{1}{n} \sum \frac{1}{y_i^2} \right) = -10 \log \left(\frac{1}{n} \sum \frac{1}{y_i^2} \right)$

Ортогональные массивы Тагути (наиболее часто используемые):

Массив	Число опытов	Макс. число факторов	Число уровней
L4	4	3	2
L8	8	7	2
L9	9	4	3
L12	12	11	2
L16	16	15	2
L18	18	8	3
L27	27	13	3

Структура эксперимента по Тагути:

text

Внутренний массив (управляемые факторы) → для каждой комбинации

↓

Внешний массив (шумовые факторы) → несколько запусков

↓

Измерение отклика для каждого сочетания шумов

↓

Расчёт SNR для данной комбинации управляемых факторов

↓

Выбор уровней управляемых факторов, максимизирующих SNR

3. Постановка задачи (ИТ-кейс)

Контекст:

Вы – Java-инженер, настраиваете JVM для высоконагруженного сервиса. Вам нужно выбрать параметры, которые **минимизируют паузы сборщика мусора (GC)** и делают их **стабильными** при разных типах нагрузки.

Управляемые факторы (внутренний массив) – 3 фактора, 3 уровня:

Фактор	Обозначение	Уровень 1 (-1)	Уровень 2 (0)	Уровень 3 (+1)
Начальный размер кучи (Xms)	A	1 ГБ	2 ГБ	4 ГБ
Максимальный размер кучи (Xmx)	B	2 ГБ	4 ГБ	8 ГБ
Тип сборщика GC	C	G1GC	ParallelGC	ZGC

Шумовые факторы (внешний массив) – 2 фактора, 2 уровня:

Шумовой фактор	Обозначение	Уровень 1 (-1)	Уровень 2 (+1)
Нагрузка (RPS)	N1	500	2000
Доля крупных объектов (>1KB)	N2	10%	40%

Отклик:

`gc_pause_ms` – средняя пауза GC (в миллисекундах). Цель: **меньше – лучше**.

Задача:

1. Построить внутренний массив L9 (управляемые факторы).
2. Для каждой комбинации управляемых факторов провести эксперимент по внешнему массиву L4 (шумовые факторы).
3. Рассчитать SNR для каждого опыта (тип "меньше – лучше").
4. Построить графики главных эффектов для SNR.
5. Выбрать оптимальные уровни управляемых факторов.
6. Провести подтверждающий эксперимент.
7. Дать рекомендации по настройке JVM.

4. Исходные данные (генерация)

Создайте симулятор JVM GC с реалистичным поведением.

```
python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from itertools import product
from statsmodels.formula.api import ols
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)

def simulate_gc_pause(xms_gb, xmx_gb, gc_type, rps, large_obj_pct):
    """
    Симуляция средней паузы GC (мс).
    xms_gb: 1,2,4
    xmx_gb: 2,4,8
    gc_type: 'G1GC', 'ParallelGC', 'ZGC'
    rps: 500, 2000
    large_obj_pct: 10, 40 (%)
    """
    # Базовое время
    base_pause = 15

    # Эффекты управляемых факторов
    # Xms: больше памяти → реже GC, но дольше пауза
    if xms_gb == 1:
        effect_xms = 5
```

```

elif xms_gb == 2:
    effect_xms = 0
else: # 4
    effect_xms = -2

# Xmx: влияет на частоту полных GC
if xmx_gb == 2:
    effect_xmx = 10
elif xmx_gb == 4:
    effect_xmx = 0
else: # 8
    effect_xmx = -3

# GC mun
gc_effects = {'G1GC': 0, 'ParallelGC': 8, 'ZGC': -5}
effect_gc = gc_effects[gc_type]

# Эффекты шумовых факторов
effect_rps = 0.01 * rps # больше нагрузки → больше паузы
effect_large = 0.3 * large_obj_pct # крупные объекты ухудшают

# Взаимодействия (важно для робастности!)
# ZGC лучше работает при высокой нагрузке
if gc_type == 'ZGC' and rps > 1000:
    interaction_zgc_rps = -8
else:
    interaction_zgc_rps = 0

# ParallelGC сильно страдает от крупных объектов
if gc_type == 'ParallelGC' and large_obj_pct > 20:
    interaction_par_large = 12
else:
    interaction_par_large = 0

# Итоговая пауза
pause = (base_pause + effect_xms + effect_xmx + effect_gc +
         effect_rps + effect_large + interaction_zgc_rps + interaction_par_large
+
         np.random.normal(0, 2)) # случайный шум

return max(0, round(pause, 1))

```

```

# Внутренний массив L9 (3 фактора, 3 уровня)
# Используем ортогональный массив Тагучи L9
inner_array = pd.DataFrame({
    'Exp': [1, 2, 3, 4, 5, 6, 7, 8, 9],
    'A (Xms)': [1, 1, 1, 2, 2, 2, 3, 3, 3],
    'B (Xmx)': [1, 2, 3, 1, 2, 3, 1, 2, 3],
    'C (GC)': [1, 2, 3, 2, 3, 1, 3, 1, 2]
})

# Преобразование уровней в натуральные значения
xms_map = {1: 1, 2: 2, 3: 4}
xmx_map = {1: 2, 2: 4, 3: 8}
gc_map = {1: 'G1GC', 2: 'ParallelGC', 3: 'ZGC'}

inner_array['A_val'] = inner_array['A (Xms)'].map(xms_map)
inner_array['B_val'] = inner_array['B (Xmx)'].map(xmx_map)
inner_array['C_val'] = inner_array['C (GC)'].map(gc_map)

# Внешний массив L4 (2 шумовых фактора, 2 уровня)
noise_array = pd.DataFrame({
    'NoiseExp': [1, 2, 3, 4],
    'N1 (RPS)': [500, 500, 2000, 2000],
    'N2 (LargeObj%)': [10, 40, 10, 40]
})

print("=== Внутренний массив L9 (управляемые факторы) ===")
print(inner_array[['Exp', 'A_val', 'B_val', 'C_val']].to_string())
print("\n=== Внешний массив L4 (шумовые факторы) ===")
print(noise_array.to_string())

# Проведение эксперимента: для каждой комбинации управляемых факторов
# запускаем все 4 комбинации шумов
results = []

for _, inner_row in inner_array.iterrows():
    exp_num = inner_row['Exp']
    xms = inner_row['A_val']
    xmx = inner_row['B_val']
    gc = inner_row['C_val']

    for _, noise_row in noise_array.iterrows():

```

```

rps = noise_row['N1 (RPS)']
large_pct = noise_row['N2 (LargeObj%)']

pause = simulate_gc_pause(xms, xmx, gc, rps, large_pct)

results.append({
    'Exp': exp_num,
    'A_Xms': xms,
    'B_Xmx': xmx,
    'C_GC': gc,
    'N1_RPS': rps,
    'N2_LargeObj': large_pct,
    'GC_Pause_ms': pause
})

df_results = pd.DataFrame(results)
print("\n=== Результаты эксперимента (36 измерений: 9x4) ===")
print(df_results.head(12))

```

5. Ход выполнения (пошаговое руководство)

Шаг 1. Импорт библиотек и подготовка данных

```

python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from itertools import product

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 5)

print(f"Всего измерений: {len(df_results)}")
print(f"Число опытов в внутреннем массиве: {df_results['Exp'].nunique()}")
print(f"Число шумовых комбинаций на опыт: {df_results.groupby('Exp').size().iloc[0]}")
)

```

Шаг 2. Расчёт SNR для каждого опыта (тип "меньше – лучше")

```
python
```

```

print("\n" + "="*60)
print("РАСЧЁТ SNR (ОТНОШЕНИЕ СИГНАЛ/ШУМ)")
print("="*60)

# Для типа "меньше – лучше": SNR = -10 * Log10( (1/n) * Σ(y_i^2) )
def calculate_snr_smaller_better(values):
    msd = np.mean(np.square(values)) # mean square deviation
    if msd <= 0:
        return 100 # идеальный случай
    return -10 * np.log10(msd)

# Расчёт SNR для каждого опыта
snr_values = []
for exp in df_results['Exp'].unique():
    exp_data = df_results[df_results['Exp'] == exp]['GC_Pause_ms'].values
    snr = calculate_snr_smaller_better(exp_data)
    snr_values.append({'Exp': exp, 'SNR': snr})

df_snr = pd.DataFrame(snr_values)
print("\nSNR для каждого опыта (чем выше, тем лучше – робастнее):")
print(df_snr)

# Добавляем SNR в данные по опытам
inner_array_with_snr = inner_array.merge(df_snr, on='Exp')
print("\nВнутренний массив с SNR:")
print(inner_array_with_snr[['Exp', 'A (Xms)', 'B (Xmx)', 'C (GC)', 'A_val', 'B_val',
'C_val', 'SNR']].to_string())

```

Шаг 3. Анализ главных эффектов (средние SNR по уровням)

```

python
print("\n" + "="*60)
print("АНАЛИЗ ГЛАВНЫХ ЭФФЕКТОВ")
print("="*60)

# Функция для расчёта среднего SNR по уровням фактора
def calculate_main_effects(df, factor_col, snr_col='SNR'):
    effects = {}
    for level in sorted(df[factor_col].unique()):
        level_data = df[df[factor_col] == level][snr_col]
        effects[f"{factor_col}={level}"] = {
            'mean': level_data.mean(),
            'std': level_data.std(),

```

```

        'count': len(level_data)
    }
    return effects

# Для каждого фактора
factors = ['A (Xms)', 'B (Xmx)', 'C (GC)']
for factor in factors:
    print(f"\n{factor}:")
    effects = calculate_main_effects(inner_array_with_snr, factor)
    for level, stats in effects.items():
        print(f"  {level}: SNR = {stats['mean']:.2f} dB (±{stats['std']:.2f})")

# Оптимальный уровень
best_level = max(effects.items(), key=lambda x: x[1]['mean'])
print(f" → Оптимальный уровень: {best_level[0]} (SNR = {best_level[1]['mean']:.2f} dB)")

# Визуализация главных эффектов
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for i, factor in enumerate(factors):
    # Группировка по уровням
    grouped = inner_array_with_snr.groupby(factor)['SNR'].mean()
    levels = grouped.index.tolist()
    means = grouped.values

    axes[i].bar(levels, means, color='steelblue')
    axes[i].set_xlabel(factor)
    axes[i].set_ylabel('Средний SNR (dB)')
    axes[i].set_title(f'Главный эффект: {factor}')
    axes[i].grid(axis='y', linestyle='--', alpha=0.7)

# Добавим значения на столбцы
for j, (lvl, val) in enumerate(zip(levels, means)):
    axes[i].text(j, val + 0.5, f'{val:.2f}', ha='center', fontsize=10)

plt.tight_layout()
plt.show()

```

Шаг 4. Выбор оптимальной комбинации

```

python
print("\n" + "="*60)

```

```

print("ВЫБОР ОПТИМАЛЬНОЙ КОМБИНАЦИИ")
print("="*60)

# Оптимальные уровни на основе главных эффектов
optimal_A = 3 # Xms = 4 ГБ (уровень 3)
optimal_B = 3 # Xmx = 8 ГБ (уровень 3)
optimal_C = 3 # ZGC (уровень 3)

print(f"Рекомендуемые уровни по главным эффектам:")
print(f"  A (Xms) = уровень {optimal_A} → {xms_map[optimal_A]} ГБ")
print(f"  B (Xmx) = уровень {optimal_B} → {mx_map[optimal_B]} ГБ")
print(f"  C (GC)  = уровень {optimal_C} → {gc_map[optimal_C]}")

# Найдём фактическую лучшую комбинацию среди проведённых опытов
best_exp = inner_array_with_snr.loc[inner_array_with_snr['SNR'].idxmax()]
print(f"\nЛучшая комбинация среди 9 опытов:")
print(f"  Опыт {best_exp['Exp']}: Xms={best_exp['A_val']} ГБ, Xmx={best_exp['B_val']} ГБ, GC={best_exp['C_val']}")
print(f"  SNR = {best_exp['SNR']:.2f} dB")

# Проверим, совпадает ли она с теоретической оптимальной
if (best_exp['A (Xms)'] == optimal_A and
    best_exp['B (Xmx)'] == optimal_B and
    best_exp['C (GC)'] == optimal_C):
    print("\n✅ Оптимальная комбинация присутствует в эксперименте!")
else:
    print("\n❌ Оптимальная комбинация отсутствует в эксперименте.")
    print("  Необходим подтверждающий эксперимент.")

```

Шаг 5. Визуализация откликов при разных шумовых условиях

```

python
print("\n" + "="*60)
print("АНАЛИЗ РОБАСТНОСТИ: ОТКЛИК ПРИ РАЗНЫХ ШУМАХ")
print("="*60)

# Для каждого опыта покажем разброс пауз GC
fig, ax = plt.subplots(figsize=(14, 6))

# Box-plot по опытам
exp_labels = [f"Exp{e}" for e in range(1, 10)]
exp_data = [df_results[df_results['Exp'] == e]['GC_Pause_ms'].values for e in range(1, 10)]

```

```

bp = ax.boxplot(exp_data, labels=exp_labels, patch_artist=True)
ax.set_xlabel('Номер опыта')
ax.set_ylabel('Пауза GC (мс)')
ax.set_title('Разброс пауз GC при различных шумовых условиях (4 комбинации шумов на о
пыт)')
ax.grid(axis='y', linestyle='--', alpha=0.3)

# Подпишем SNR над каждым боксплотом
for i, exp in enumerate(range(1, 10)):
    snr_val = df_snr[df_snr['Exp'] == exp]['SNR'].values[0]
    ax.text(i+1, ax.get_ylim()[1] - 2, f'SNR={snr_val:.1f}',
           ha='center', fontsize=9, bbox=dict(boxstyle='round', facecolor='wheat', a
lpha=0.5))

plt.tight_layout()
plt.show()

# Покажем лучший и худший опыт
best_exp_num = df_snr.loc[df_snr['SNR'].idxmax(), 'Exp']
worst_exp_num = df_snr.loc[df_snr['SNR'].idxmin(), 'Exp']

print(f"\nЛучший опыт (Exp {best_exp_num}):")
best_data = df_results[df_results['Exp'] == best_exp_num]
print(best_data[['A_Xms', 'B_Xmx', 'C_GC', 'N1_RPS', 'N2_LargeObj', 'GC_Pause_ms']].t
o_string())
print(f"    SNR = {df_snr[df_snr['Exp'] == best_exp_num]['SNR'].values[0]:.2f} dB")

print(f"\nХудший опыт (Exp {worst_exp_num}):")
worst_data = df_results[df_results['Exp'] == worst_exp_num]
print(worst_data[['A_Xms', 'B_Xmx', 'C_GC', 'N1_RPS', 'N2_LargeObj', 'GC_Pause_ms']].
to_string())
print(f"    SNR = {df_snr[df_snr['Exp'] == worst_exp_num]['SNR'].values[0]:.2f} dB")

```

Шаг 6. Подтверждающий эксперимент

```

python
print("\n" + "="*60)
print("ПОДТВЕРЖДАЮЩИЙ ЭКСПЕРИМЕНТ")
print("="*60)

# Оптимальная конфигурация
optimal_config = {

```

```

    'A_Xms': xms_map[optimal_A],
    'B_Xmx': xmx_map[optimal_B],
    'C_GC': gc_map[optimal_C]
}

print(f"Тестируемая конфигурация: Xms={optimal_config['A_Xms']} ГБ, "
      f"Xmx={optimal_config['B_Xmx']} ГБ, GC={optimal_config['C_GC']}")

# Проведём подтверждающий эксперимент (прогон по всем 4 шумовым условиям)
confirm_results = []
for _, noise_row in noise_array.iterrows():
    rps = noise_row['N1 (RPS)']
    large_pct = noise_row['N2 (LargeObj%)']

    # Несколько повторений для оценки стабильности
    repeats = []
    for _ in range(3):
        pause = simulate_gc_pause(
            optimal_config['A_Xms'], optimal_config['B_Xmx'],
            optimal_config['C_GC'], rps, large_pct
        )
        repeats.append(pause)

    for p in repeats:
        confirm_results.append({
            'N1_RPS': rps,
            'N2_LargeObj': large_pct,
            'GC_Pause_ms': p
        })

df_confirm = pd.DataFrame(confirm_results)
print("\nРезультаты подтверждающего эксперимента (12 измерений):")
print(df_confirm)

# Расчёт SNR для подтверждающего эксперимента
confirm_snr = calculate_snr_smaller_better(df_confirm['GC_Pause_ms'].values)
print(f"\nSNR подтверждающей конфигурации: {confirm_snr:.2f} dB")

# Сравнение с лучшим опытом из исходного эксперимента
best_snr = df_snr['SNR'].max()
print(f"Лучший SNR из исходного эксперимента: {best_snr:.2f} dB")

```

```
if confirm_snr >= best_snr:
    print("\n✅ Подтверждающий эксперимент подтвердил робастность оптимальной конфигурации!")
else:
    print("\n❌ Подтверждающий эксперимент показал результат хуже ожидаемого.")
    print("    Возможно, требуется дополнительная оптимизация.")
```

Шаг 7. Сравнительный анализ: робастная vs неробастная конфигурация

python

Выберем неробастную конфигурацию для сравнения (например, худший опыт)

```
worst_config = {
    'A_Xms': worst_data['A_Xms'].iloc[0],
    'B_Xmx': worst_data['B_Xmx'].iloc[0],
    'C_GC': worst_data['C_GC'].iloc[0]
}

print("\n" + "="*60)
print("СРАВНЕНИЕ: РОБАСТНАЯ vs НЕРОБАСТНАЯ КОНФИГУРАЦИЯ")
print("="*60)

print(f"Робастная конфигурация: Xms={optimal_config['A_Xms']} ГБ, "
      f"Xmx={optimal_config['B_Xmx']} ГБ, GC={optimal_config['C_GC']}")
print(f"Неробастная конфигурация: Xms={worst_config['A_Xms']} ГБ, "
      f"Xmx={worst_config['B_Xmx']} ГБ, GC={worst_config['C_GC']}")

# Собираем данные для сравнения
robust_pauses = df_confirm['GC_Pause_ms'].values
nonrobust_pauses = worst_data['GC_Pause_ms'].values
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
```

Гистограммы

```
axes[0].hist(robust_pauses, bins=8, alpha=0.7, label='Робастная', color='green')
axes[0].hist(nonrobust_pauses, bins=8, alpha=0.7, label='Неробастная', color='red')
axes[0].set_xlabel('Пауза GC (мс)')
axes[0].set_ylabel('Частота')
axes[0].set_title('Распределение пауз GC')
axes[0].legend()
```

```
# Box-plot
```

```
axes[1].boxplot([robust_pauses, nonrobust_pauses],
                labels=['Робастная', 'Неробастная'])
```

```

axes[1].set_ylabel('Пауза GC (мс)')
axes[1].set_title('Сравнение разброса пауз')
axes[1].grid(axis='y', linestyle='--', alpha=0.3)

plt.tight_layout()
plt.show()

# Статистическое сравнение
print(f"\nСтатистика:")
print(f"  Робастная:      среднее = {np.mean(robust_pauses):.2f} мс, "
      f"ст.откл. = {np.std(robust_pauses):.2f} мс")
print(f"  Неробастная:   среднее = {np.mean(nonrobust_pauses):.2f} мс, "
      f"ст.откл. = {np.std(nonrobust_pauses):.2f} мс")
print(f"  Улучшение среднего: {(np.mean(nonrobust_pauses) - np.mean(robust_pauses)):.2f} мс")
print(f"  Улучшение стабильности: {(np.std(nonrobust_pauses) - np.std(robust_pauses)):.2f} мс")

# t-test
t_stat, t_p = stats.ttest_ind(robust_pauses, nonrobust_pauses)
print(f"  t-test: p = {t_p:.4f} → {'различия значимы' if t_p < 0.05 else 'различия не значимы'}")

```

Шаг 8. Анализ взаимодействий (опционально)

```

python
print("\n" + "="*60)
print("АНАЛИЗ ВЗАИМОДЕЙСТВИЙ (ДЛЯ ОПТИМИЗАЦИИ)")
print("="*60)

# Построим матрицу средних пауз для пар факторов
pivot_ab = df_results.pivot_table(
    values='GC_Pause_ms',
    index='A_Xms',
    columns='B_Xmx',
    aggfunc='mean'
)

print("Средние паузы GC (мс) в зависимости от A (Xms) и B (Xmx):")
print(pivot_ab.round(1))

# График взаимодействия A×B
plt.figure(figsize=(8, 5))

```

```

for b in pivot_ab.columns:
    plt.plot(pivot_ab.index, pivot_ab[b], marker='o', label=f'Xmx={b} ГБ')
plt.xlabel('Xms (ГБ)')
plt.ylabel('Средняя пауза GC (мс)')
plt.title('Взаимодействие: Xms x Xmx')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.3)
plt.show()

# Анализ: для ZGC (лучшего GC) проверим чувствительность к шумам
zgc_data = df_results[df_results['C_GC'] == 'ZGC']
print("\nZGC: чувствительность к шумам:")
print(zgc_data.groupby(['N1_RPS', 'N2_LargeObj'])['GC_Pause_ms'].agg(['mean', 'std'])
.round(1))

```

Шаг 9. Итоговый вывод и рекомендации

```

python
print("\n" + "="*60)
print("ИТОГОВЫЙ ВЫВОД И РЕКОМЕНДАЦИИ")
print("="*60)

print("РЕЗУЛЬТАТЫ ПАРАМЕТРИЧЕСКОЙ ОПТИМИЗАЦИИ (МЕТОД ТАГУЧИ):")
print("-" * 50)

print(f"\n1. Оптимальная конфигурация JVM:")
print(f"    • Xms (начальная куча) = {optimal_config['A_Xms']} ГБ")
print(f"    • Xmx (максимальная куча) = {optimal_config['B_Xmx']} ГБ")
print(f"    • Сборщик мусора = {optimal_config['C_GC']}")

print(f"\n2. Показатели робастности:")
print(f"    • SNR = {confirm_snr:.2f} dB")
print(f"    • Средняя пауза GC = {np.mean(robust_pauses):.2f} мс")
print(f"    • Стандартное отклонение = {np.std(robust_pauses):.2f} мс")

print(f"\n3. Сравнение с неробастной конфигурацией:")
print(f"    • Улучшение среднего времени: {(np.mean(nonrobust_pauses) - np.mean(robust
_pauses)):.1f} мс")
print(f"    • Улучшение стабильности: {(np.std(nonrobust_pauses) - np.std(robust_pause
s)):.1f} мс")

print(f"\n4. Рекомендации:")
print(f"    • Внедрить оптимальную конфигурацию в production.")

```

```

print("    • Продолжить мониторинг GC-пауз в течение 2 недель.")
print("    • При изменении паттерна нагрузки (RPS > 3000) повторить эксперимент.")
print("    • Рассмотреть возможность использования ZGC для high-load сценариев.")

# Сохранение результатов
summary_df = pd.DataFrame({
    'Конфигурация': ['Робастная (оптимальная)', 'Неробастная (худшая)'],
    'Xms (ГБ)': [optimal_config['A_Xms'], worst_config['A_Xms']],
    'Xmx (ГБ)': [optimal_config['B_Xmx'], worst_config['B_Xmx']],
    'GC': [optimal_config['C_GC'], worst_config['C_GC']],
    'Средняя пауза (мс)': [np.mean(robust_pauses), np.mean(nonrobust_pauses)],
    'Ст.откл. (мс)': [np.std(robust_pauses), np.std(nonrobust_pauses)],
    'SNR (дБ)': [confirm_snr, df_snr[df_snr['Exp'] == worst_exp_num]['SNR'].values[0]]
})
print("\n5. Сводная таблица результатов:")
print(summary_df.round(2))

```

6. Индивидуальные задания (6 вариантов)

Каждый студент получает **один из вариантов**. Необходимо выполнить полный анализ по шагам 1–9.

Вар.	Система	Управляемые факторы (3 фактора, 3 уровня)	Шумовые факторы (2 фактора, 2 уровня)	Отклик (цель)
1	JVM GC (как в примере)	Xms (1/2/4 ГБ), Xmx (2/4/8 ГБ), GC (G1/Parallel/ZGC)	RPS (500/2000), крупные объекты (10/40%)	Пауза GC (меньше)
2	Кэш Redis	maxmemory (256/512/1024 МБ), policy (LRU/LFU/TTL), save_interval (60/300/600 с)	RPS (1000/5000), размер объекта (1/100 КБ)	Hit rate (больше)
3	Алгоритм сжатия	уровень (1/5/9), блок (4/64/256)	тип данных (текст/бинарный),	Время+размер (меньше)

Вар.	Система	Управляемые факторы (3 фактора, 3 уровня)	Шумовые факторы (2 фактора, 2 уровня)	Отклик (цель)
		КБ), словарь (off/on/auto)	энтропия (низкая/высокая)	
4	Балансировщик	алгоритм (round-robin/least-conn/ip-hash), таймаут (10/30/60 с), буфер (4/16/64 КБ)	RPS (500/3000), доля долгих запросов (5/25%)	Время ответа (меньше)
5	PostgreSQL	shared_buffers (128/512/2048 МБ), work_mem (4/16/64 МБ), effective_cache_size (256/1024/4096 МБ)	число соединений (10/100), сложность запросов (простой/сложный)	Время запроса (меньше)
6	Автомасштабирование	порог CPU (50/70/90%), шаг масштабирования (1/2/4 инстанса), период оценки (60/300/600 с)	скорость роста нагрузки (медленно/быстро), длительность пика (короткий/длинный)	Время реакции (меньше)

7. Структура отчета

1. **Цель работы**
2. **Описание управляемых и шумовых факторов**
3. **Матрица планирования** (внутренний и внешний массивы)
4. **Расчёт SNR для каждого опыта**
5. **Анализ главных эффектов** (таблицы, графики)
6. **Выбор оптимальной комбинации**
7. **Подтверждающий эксперимент** и его результаты

8. Сравнение робастной и неробастной конфигураций
9. Выводы и рекомендации
10. Ответы на контрольные вопросы

8. Контрольные вопросы (для защиты)

1. В чём основная идея метода Тагучи?
2. Что такое отношение сигнал/шум (SNR) и как оно рассчитывается для разных типов задач?
3. Чем отличаются управляемые факторы от шумовых?
4. Что такое ортогональный массив и зачем он нужен?
5. Какова структура эксперимента по Тагучи (внутренний и внешний массивы)?
6. Что такое функция потерь Тагучи и почему она квадратичная?
7. Как интерпретировать SNR: чем выше – тем лучше?
8. Зачем нужен подтверждающий эксперимент?
9. В чём преимущество робастной конфигурации перед обычной оптимальной?
10. Каковы ограничения метода Тагучи?

9. Полный код в одном блоке (для копирования)

```
python
# Полный код метода Тагучи для оптимизации JVM

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats

np.random.seed(42)

# Симулятор GC
def gc_pause(xms, xmx, gc, rps, large_pct):
    base = 15
    effect = 0
    if xms == 1: effect += 5
    elif xms == 4: effect -= 2
    if xmx == 2: effect += 10
    elif xmx == 8: effect -= 3
    effect += {'G1GC':0, 'ParallelGC':8, 'ZGC':-5}[gc]
    effect += 0.01 * rps + 0.3 * large_pct
    if gc == 'ZGC' and rps > 1000: effect -= 8
    if gc == 'ParallelGC' and large_pct > 20: effect += 12
    return max(0, base + effect + np.random.normal(0, 2))
```

```

# Внутренний массив L9
inner = pd.DataFrame({
    'A': [1,1,1,2,2,2,3,3,3],
    'B': [1,2,3,1,2,3,1,2,3],
    'C': [1,2,3,2,3,1,3,1,2]
})
xms_map = {1:1,2:2,3:4}
xmx_map = {1:2,2:4,3:8}
gc_map = {1:'G1GC',2:'ParallelGC',3:'ZGC'}

# Внешний массив L4
noise = [(500,10), (500,40), (2000,10), (2000,40)]

# Эксперимент
results = []
for _, row in inner.iterrows():
    for rps, large in noise:
        pause = gc_pause(xms_map[row['A']], xmx_map[row['B']], gc_map[row['C']], rps,
            large)
        results.append({'Exp': row.name+1, 'A':row['A'], 'B':row['B'], 'C':row['C'],
            'RPS':rps, 'Large':large, 'Pause':pause})
df = pd.DataFrame(results)

# Расчёт SNR
snr = []
for exp in range(1,10):
    pauses = df[df['Exp']==exp]['Pause'].values
    msd = np.mean(pauses**2)
    snr.append(-10*np.log10(msd))
df_snr = pd.DataFrame({'Exp':range(1,10), 'SNR':snr})

# Главные эффекты
for factor, name in [('A','Xms'), ('B','Xmx'), ('C','GC')]:
    print(f"\n{name}:")
    for lvl in [1,2,3]:
        mean_snr = df_snr.merge(inner, left_on='Exp', right_index=True)[inner[factor]
            ==lvl]['SNR'].mean()
        print(f"    Уровень {lvl}: SNR = {mean_snr:.2f}")

print(f"\nОптимум: A=3, B=3, C=3 → Xms=4ГБ, Xmx=8ГБ, GC=ZGC")

```

Данная лабораторная работа даёт практические навыки робастного проектирования, которые позволяют создавать программные системы, сохраняющие стабильность работы в реальных, изменчивых условиях эксплуатации.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по организации и проведению самостоятельной работы по дисциплине
«Проектирование и анализ экспериментов» для студентов
направления подготовки
09.04.04 Программная инженерия Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом
инженерных систем»

Ставрополь 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ	7
2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА.....	11
3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ	12
4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ	12
5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ.....	13

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Формирование у магистрантов компетенций в области методологии планирования экспериментов, сбора и статистического анализа данных для оценки характеристик сложных программных систем и принятия обоснованных инженерных решений.

Задачи освоения дисциплины:

- освоение методов дисперсионного и регрессионного анализа;
- изучение факторного планирования;
- приобретение навыков использования инструментов (R, Python, MATLAB) для анализа экспериментальных данных в сфере ИТ.

2. Место дисциплины (модуля) в структуре образовательной программы

Дисциплина «Проектирование и анализ экспериментов» относится к дисциплинам обязательной части.

3. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
УК-1. Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий	ИД-1 _{УК-1} осуществляет разработку стратегии действий для выявления и решения проблемной ситуации.	осуществляет разработку стратегии действий для выявления и решения проблемной ситуации.
	ИД-2 _{УК-1} вырабатывает стратегию действий, принимает конкретные решения для ее реализации.	вырабатывает стратегию действий, принимает конкретные решения для ее реализации.
	ИД-3 _{УК-1} ставит цели, определяет способы ее достижения, разрабатывает стратегий действий.	ставит цели, определяет способы ее достижения, разрабатывает стратегий действий.
ОПК-1. Способен самостоятельно приобретать, развивать и применять математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте	ИД-1 _{ОПК-1} . приобретать математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте;	приобретает математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте
	ИД-2 _{ОПК-1} . развивать математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в	развивает математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в

	междисциплинарном контексте	междисциплинарном контексте
	ИД-3 ОПК-1. применять математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте	применяет математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте
ОПК-5. Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем	ИД-1 ОПК-5. разрабатывать программное и аппаратное обеспечение информационных и автоматизированных систем	Разрабатывает программное и аппаратное обеспечение информационных и автоматизированных систем
	ИД-2 ОПК-5. модернизировать программное и аппаратное обеспечение информационных и автоматизированных систем	Модернизирует программное и аппаратное обеспечение информационных и автоматизированных систем

4. Объем учебной дисциплины и формы контроля *

Объем занятий: всего: 5 з.е. 180 ак.ч.	ОФО, в астр. часах
Контактная работа:	54
Лекции/из них практическая подготовка	18
Лабораторных работ/из них практическая подготовка	36
Практических занятий/из них практическая подготовка	
Самостоятельная работа	126
Формы контроля	
Экзамен	
Зачет	
Зачет с оценкой	
Расчетно-графические работы	
Курсовые работы	
Контрольные работы	

* Дисциплина (модуль) предусматривает применение электронного обучения, дистанционных образовательных технологий

5. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием количества часов и видов занятий

№	Раздел (тема) дисциплины и краткое содержание	Формируемые компетенции, индикаторы	очная форма			
			Контактная работа обучающихся с преподавателем /из них в форме практической подготовки, часов			Самостоятельная работа, часов
			Лекции	Практические занятия	Лабораторные работы	
1	Раздел 1. Введение в планирование эксперимента. Основные понятия: фактор, отклик, воспроизводимость, рандомизация. Этапы постановки вычислительного эксперимента. Классификация задач оптимизации в программной инженерии.	УК-1; ОПК-1; ОПК-5	2		6	21
2	Раздел 2. Статистическая обработка результатов. Проверка нормальности распределения (критерии Колмогорова-Смирнова, Шапиро-Уилка). Проверка гипотез о средних и дисперсиях. Оценка погрешности измерений.	УК-1; ОПК-1; ОПК-5	4		6	21
3	Раздел 3. Однофакторный и многофакторный дисперсионный анализ. ANOVA для зависимых и независимых выборок. Post-hoc анализ (Тьюки, Шеффе). Применение ANOVA для сравнения производительности нескольких алгоритмов или версий ПО.	УК-1; ОПК-1; ОПК-5	2		6	21

4	Раздел 4. Регрессионный анализ. Построение линейных и полиномиальных моделей. Метод наименьших квадратов. Полный факторный эксперимент (ПФЭ 2^k). Дробный факторный эксперимент. Регрессионный анализ для метрик качества ПО (например, предсказание времени выполнения от объема данных).	УК-1; ОПК-1; ОПК-5	4		6	21
5	Раздел 5. Методы оптимизации эксперимента. Крутое восхождение (метод Бокса-Уилсона). Симплексный метод планирования. Робастное проектирование (Taguchi methods) для повышения надежности ПО.	УК-1; ОПК-1; ОПК-5	4		6	21
6	Раздел 6. Специфика экспериментов в ИТ. Дизайн А/Б тестов в веб-разработке. Анализ логов и нагрузочное тестирование. Методы Монте-Карло для оценки рисков.	УК-1; ОПК-1; ОПК-5	2		6	21
	ИТОГО за 3 семестр		18		36	126
	ИТОГО		18		36	126

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Все виды самостоятельной работы по дисциплине "Проектирование и анализ экспериментов" приведены в таблице «Технологическая карта самостоятельной работы». В основу самостоятельной работы студентов по дисциплине положено конспектирование лекций и рекомендуемых источников, подготовка к компьютерному тестированию знаний, оформление и подготовка к защите отчетов по лабораторным работам, выполнению разноуровневых индивидуальных заданий в ходе лабораторных работ. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности, которые студенты предъявляют преподавателю в ходе лабораторных занятий или в часы индивидуальных консультаций.

Таблица 1 - Технологическая карта самостоятельной работы.

Код оцениваемой компетенции, индикаторов	Этап формирования компетенции (№ темы) (в соответствии с рабочей программой дисциплины)	Средства и технологии оценки	Вид контроля, аттестация (текущий/промежуточный)	Тип контроля (устный, письменный или с использованием технических средств)	Наименование оценочного средства
1 семестр					
УК-1.	Темы № 1–2	Собеседование, оценка ответов по вопросам заданных тем	текущий	устный	вопросы для собеседования
ОПК-1.	Темы № 3–4	Собеседование, оценка ответов по	текущий	устный	вопросы для собеседования

		вопросам заданных тем			
ОПК-5.	Темы № 5–6	Собеседование, оценка ответов по вопросам заданных тем	текущий	устный	вопросы для собеседования

Для успешного освоения дисциплины, необходимо выполнить указанные в таблице 1 виды самостоятельной работы, используя рекомендуемые источники информации.

8.1. Перечень основной и дополнительной литературы, необходимой для освоения дисциплины (модуля)

8.1.1. Перечень основной литературы:

1. Глухих В. В. (Виктор Владимирович), Шкуро А. Е. (Алексей Евгеньевич), Артемов А. В. (Артем Вячеславович) и др. Математическое планирование экспериментов и анализ их результатов с применением компьютерных программ. учебное пособие. 2023г.

2. Гефан Г. Д., Ширяева Н. К. «Основы теории эксперимента» учебное пособие/ Гефан Г. Д., Ширяева Н. К. – Иркутск: ИрГУПС, 2017. – 136 с.

8.1.2. Перечень дополнительной литературы:

1. Меледина Т.В., Данина М.М. Методы планирования и обработки результатов научных исследований: Учеб. пособие. – СПб.: НИУ ИТМО; ИХиБТ, 2015. – 110 с.

8.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине

8.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

www.biblioclub.ru - Электронная библиотечная система «Университетская библиотека онлайн»

www.eLibrary.ru - Научная электронная библиотека

www.iprbookshop.ru - Электронно-библиотечная система IPRbooks

Особенности освоения дисциплины (модуля) лицами с ограниченными возможностями здоровья

Обучающимся с ограниченными возможностями здоровья предоставляются специальные учебники, учебные пособия и дидактические материалы, специальные технические средства обучения коллективного и индивидуального пользования, услуги ассистента (помощника), оказывающего

обучающимся необходимую техническую помощь, а также услуги сурдопереводчиков и тифлосурдопереводчиков.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья может быть организовано совместно с другими обучающимися, а также в отдельных группах.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья осуществляется с учетом особенностей психофизического развития, индивидуальных возможностей и состояния здоровья.

В целях доступности получения высшего образования по образовательной программе лицами с ограниченными возможностями здоровья при освоении дисциплины (модуля) обеспечивается:

1) для лиц с ограниченными возможностями здоровья по зрению:

- присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочесть и оформить задание, в том числе, записывая под диктовку),

- письменные задания, а также инструкции о порядке их выполнения оформляются увеличенным шрифтом,

- специальные учебники, учебные пособия и дидактические материалы (имеющие крупный шрифт или аудиофайлы),

- индивидуальное равномерное освещение не менее 300 люкс,
- при необходимости студенту для выполнения задания предоставляется увеличивающее устройство;
- 2) для лиц с ограниченными возможностями здоровья по слуху:
 - присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочитать и оформить задание, в том числе, записывая под диктовку),
 - обеспечивается наличие звукоусиливающей аппаратуры коллективного пользования, при необходимости обучающемуся предоставляется звукоусиливающая аппаратура индивидуального пользования;
 - обеспечивается надлежащими звуковыми средствами воспроизведения информации;
- 3) для лиц с ограниченными возможностями здоровья, имеющих нарушения опорно-двигательного аппарата (в том числе с тяжелыми нарушениями двигательных функций верхних конечностей или отсутствием верхних конечностей):
 - письменные задания выполняются на компьютере со специализированным программным обеспечением или надиктовываются ассистенту;
 - по желанию студента задания могут выполняться в устной форме.

Особенности реализации дисциплины с применением дистанционных образовательных технологий и электронного обучения

Согласно части 1 статьи 16 Федерального закона от 29 декабря 2012 г. № 273-ФЗ «Об образовании в Российской Федерации» под *электронным обучением* понимается организация образовательной деятельности с применением содержащейся в базах данных и используемой при реализации образовательных программ информации и обеспечивающих ее обработку информационных технологий, технических средств, а также информационно-телекоммуникационных сетей, обеспечивающих передачу по линиям связи указанной информации, взаимодействие обучающихся и педагогических работников. Под *дистанционными образовательными технологиями* понимаются образовательные технологии, реализуемые в основном с применением информационно-телекоммуникационных сетей при опосредованном (на расстоянии) взаимодействии обучающихся и педагогических работников.

Реализация дисциплины может быть осуществлена с применением дистанционных образовательных технологий и электронного обучения полностью или частично. Компоненты УМК дисциплины (рабочая программа дисциплины, оценочные и методические материалы, формы аттестации), реализуемой с применением дистанционных образовательных технологий и электронного обучения, содержат указание на их использование.

При организации образовательной деятельности с применением дистанционных образовательных технологий и электронного обучения могут предусматриваться асинхронный и синхронный способы осуществления взаимодействия участников образовательных отношений посредством информационно-телекоммуникационной сети «Интернет».

При применении дистанционных образовательных технологий и электронного обучения в

расписании по дисциплине указываются: способы осуществления взаимодействия участников образовательных отношений посредством информационно-телекоммуникационной сети «Интернет» (ВКС-видеоконференцсвязь, ЭТ – электронное тестирование); ссылки на электронную информационно-образовательную среду СКФУ, на образовательные платформы и ресурсы иных организаций, к которым предоставляется открытый доступ через информационно-телекоммуникационную сеть

«Интернет»; для синхронного обучения - время проведения онлайн-занятий и преподаватели; для асинхронного обучения - авторы онлайн-курсов.

При организации промежуточной аттестации с применением дистанционных образовательных технологий и электронного обучения используются Методические рекомендации по применению технических средств, обеспечивающих объективность результатов при проведении промежуточной и государственной итоговой аттестации по образовательным программам высшего образования - программам бакалавриата, программам специалитета и программам магистратуры с применением дистанционных образовательных технологий (Письмо Минобрнауки России от 07.12.2020 г. № МН- 19/1573-АН "О направлении

методических рекомендаций").

Реализация дисциплины с применением электронного обучения и дистанционных образовательных технологий осуществляется с использованием электронной информационно-образовательной среды СКФУ, к которой обеспечен доступ обучающихся через информационно-телекоммуникационную сеть «Интернет», или с использованием ресурсов иных организаций, в том числе платформ, предоставляющих сервисы для проведения видеоконференций, онлайн-встреч и дистанционного обучения (МТС-Линк), а также с использованием возможностей социальных сетей для осуществления коммуникации обучающихся и преподавателей.

Учебно-методическое обеспечение дисциплины, реализуемой с применением электронного обучения и дистанционных образовательных технологий, включает представленные в электронном виде рабочую программу, учебно-методические пособия или курс лекций, методические указания к выполнению различных видов учебной деятельности обучающихся, предусмотренных дисциплиной, и прочие учебно-методические материалы, размещенные в информационно-образовательной среде СКФУ.

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

При самостоятельном изучении тем лекционных занятий и заданной литературы студентам рекомендуется:

- в день проведения занятий проработать, изученный на занятии учебный материал по конспекту. При этом необходимо выделить вопросы, на которые следует обратить большее внимание при дальнейшем изучении текущей темы. После проработки учебного материала необходимо ответить на, рекомендуемые контрольные вопросы;
- с целью качественного закрепления изученного материала в последующем следует более детально проработать учебный материал с использованием рекомендованной литературы. Студентам рекомендуется выделить вопросы, требующие более детальной проработки с помощью преподавателя;
- накануне проведения лекции студенту рекомендуется закрепить ранее изученный теоретический материал, подготовить вопросы, требующие уточнения у преподавателя.

Рекомендации по организации работы с литературой

Для овладения, закрепления и систематизации знаний студентам рекомендуется самостоятельная работа с рекомендованной литературой и конспектом лекций. При самостоятельной работе с рекомендованной литературой студентам рекомендуется проводить конспектирование учебного материала, изложенного в рекомендованных источниках.

Конспектирование источников проводится при детальном изучении темы, при этом студентам рекомендуется:

- дополнять конспект лекционного занятия путем его расширения по наиболее важным вопросам, рассмотренным на занятии. Конспектирование источников заключается в дополнении конспекта следующими положениями:
 - основные определения и их физический смысл;
 - основные математические соотношения с раскрытием их физического смысла;
 - выводы по учебным вопросам, изучаемой темы занятий.

Контроль этого вида самостоятельной работы осуществляется на учебных занятиях путем проверки преподавателем конспекта лекций и собеседования.

При конспектировании источников студенты должны исходить из рационального объема, конспектируемого материала. Конспект должен быть кратким и понятным при его использовании после изучения текущей темы.

При конспектировании источников рекомендуется отдельные рисунки, наиболее важные их фрагменты, наиболее важные формулы выделять цветом. При ведении конспекта на занятии рекомендуется заполнять две трети страницы листа тетради по вертикали, а свободную часть использовать при дополнении конспекта, по изучаемому вопросу.

Для успешного освоения дисциплины, необходимо самостоятельно детально изучить представленные темы по рекомендуемым источникам информации.

Конспектирование источников оценивается удовлетворительно и неудовлетворительно. Работа по конспектированию источников оценивается удовлетворительно если:

- конспект имеет достаточный объем детализации по, изучаемой теме, обеспечивающий заданный уровень освоения теоретического материала дисциплины;
- конспект оформлен аккуратно, изучаемый материал изложен последовательно в соответствии с порядком изучения дисциплины, содержит обобщающие выводы по каждой теме.

Работа по конспектированию источников оценивается неудовлетворительно, если не выполнены требования на оценку удовлетворительно.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

При выполнении практикума по дисциплине для достижения базового уровня сформированности компетенций студент должен уметь применять базовые знания по разделам дисциплины, владеть навыками самостоятельного решения учебных задач базового уровня.

При выполнении практикума по дисциплине для достижения повышенного уровня сформированности компетенций студент должен уметь применять повышенные знания по разделам дисциплины, владеть навыками самостоятельного решения учебных задач повышенного уровня.

Защита практической работы осуществляется путем собеседования студента с преподавателем. При собеседовании студент представляет на проверку отчет по практической работе. Собеседование со студентом, претендующим на оценку «отлично» проводится по вопросам и практическим заданиям повышенного уровня обучения. Собеседование со студентом, не претендующим на оценку «отлично» проводится по вопросам и практическим заданиям базового уровня обучения.

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ

ПО ДИСЦИПЛИНЕ Управление жизненным циклом инженерных систем

Базовый уровень

1. Основные понятия: фактор, отклик, воспроизводимость, рандомизация.
2. Этапы постановки вычислительного эксперимента.
3. Классификация задач оптимизации в программной инженерии.
4. Проверка нормальности распределения (критерии Колмогорова-Смирнова, Шапиро-Уилка).
5. Проверка гипотез о средних и дисперсиях. Оценка погрешности измерений.
6. ANOVA для зависимых и независимых выборок.
7. Post-hoc анализ (Тьюки, Шеффе).
8. Применение ANOVA для сравнения производительности нескольких алгоритмов или версий ПО.
9. Построение линейных и полиномиальных моделей.
10. Метод наименьших квадратов.

Повышенный уровень

1. Полный факторный эксперимент (ПФЭ 2^k).

2. Дробный факторный эксперимент.
3. Регрессионный анализ для метрик качества ПО (например, предсказание времени выполнения от объема данных).
4. Крутое восхождение (метод Бокса-Уилсона).
5. Симплексный метод планирования.
6. Робастное проектирование (Taguchi methods) для повышения надежности ПО.
7. Дизайн А/Б тестов в веб-разработке.
8. Анализ логов и нагрузочное тестирование.
9. Методы Монте-Карло для оценки рисков.

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически стройно его излагает, умеет тесно увязывать теорию с практикой, свободно справляется с вопросами не только базового, но и повышенного уровня, причем не затрудняется с ответом при видоизменении заданий, правильно обосновывает принятое решение.

Оценка «хорошо» выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы базового уровня, правильно применяет теоретические положения при решении практических вопросов базового уровня, владеет необходимыми навыками и приемами их выполнения.

Оценка «удовлетворительно» выставляется студенту, если он имеет знания только основного материала базового уровня, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала.

Оценка «неудовлетворительно» выставляется студенту, не знающему значительной части программного материала, допускающему грубые ошибки.

Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

Процедура проведения данного оценочного мероприятия включает в себя собеседование.

Предлагаемые студенту задания позволяют проверить компетенции УК-1; ОПК-1, ОПК-5. При проверке задания оценивается:

- объем и глубина усвоенного программного материала;
- последовательность, четкость и логическая стройность его изложения.