Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)«
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *ИУК «Информатика и управление»*

КАФЕДРА *ИУК4 «Программное обеспечение ЭВМ, информационные технологии»*

ЛАБОРАТОРНАЯ РАБОТА №1.2

«Линейное программирование. Графический метод»

ДИСЦИПЛИНА: «Методы принятия решений в программной инженерии»

Выполнил: студент гр. ИУК4-72Б __________________ (Моряков В.Ю.)
(Подпись) (Ф.И.О.)

Проверил: __________________ (Никитенко У.В.)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Бальная оценка:
- Оценка:

Калуга, 2025

Цель работы: Ознакомиться с графическим методом решения задач линейного программирования.

Задачи

1) Составить математическую модель:
2) Описать переменные, параметры модели
3) Составить целевую функцию и ограничения.
4) Применить графический метод для решения задачи.
5) Провести анализ чувствительность к исходным данным.

Вариант 10

Вариант 10

Для производства двух видов автомобильных деталей А и Б предприятие использует 3 вида сырья. Другие условия задачи приведены в таблице.

Вид сырья	Нормы расхода сырья на одну деталь, кг		Общее количество сырья, кг
	A	B	
I	12	4	300
II	4	4	120
III	3	12	252
Прибыль от реализации одной детали, ден. ед.	30	40	

Составить такой план выпуска продукции, при котором прибыль от реализации продукции будет максимальна, при условии, что изделий Б нужно выпустить не менее, чем изделий А.

Определите стоимость единицы изменения граничных значений ежедневного выпуска деталей А и В.

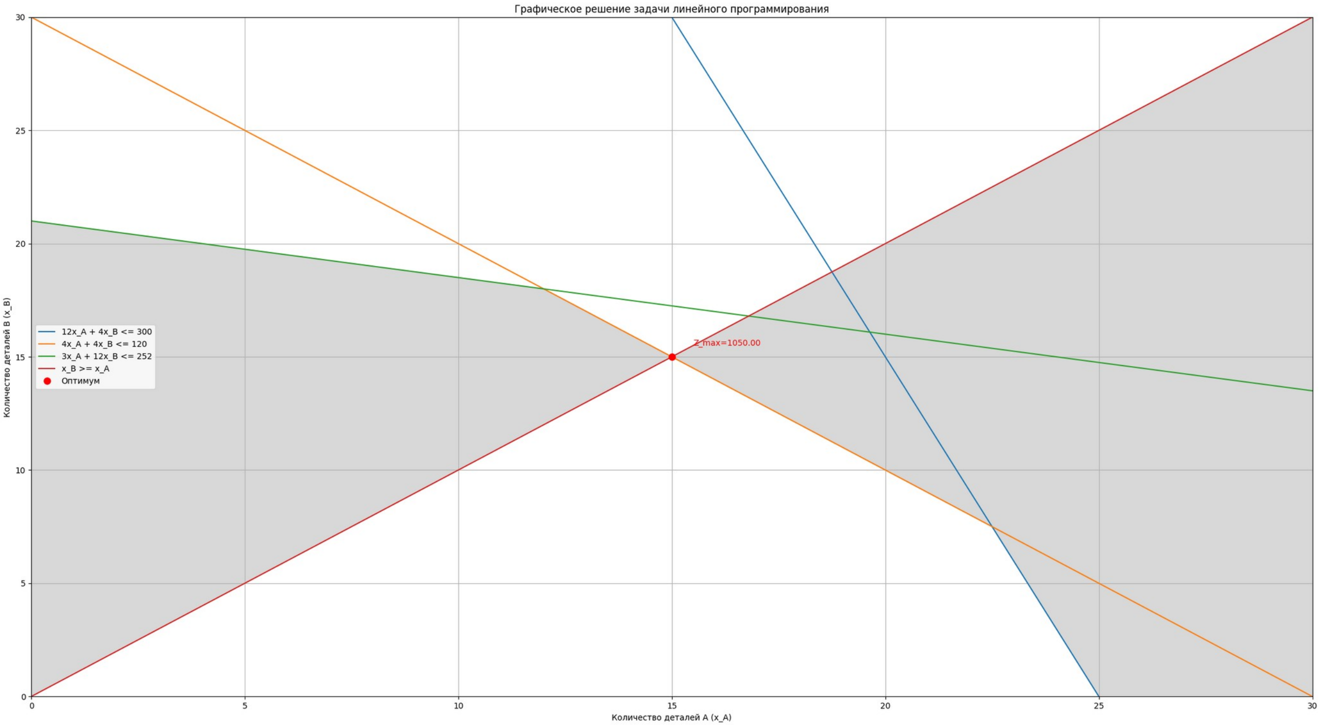
Результаты работы

1. Составить математическую модель — код приведен в листинге
2. Описать переменные, параметры модели

```
=== Переменные модели ===
x_A: количество деталей типа A, выпускаемых в день (x_A >= 0)
x_B: количество деталей типа B, выпускаемых в день (x_B >= 0)
Особое условие: x_B >= x_A (деталей B нужно выпускать не меньше, чем A)

=== Параметры модели ===
Сырьё I: a_I_A = 12 кг на деталь A, a_I_B = 4 кг на деталь B, доступно S_I = 300 кг
Сырьё II: a_II_A = 4 кг на деталь A, a_II_B = 4 кг на деталь B, доступно S_II = 120 кг
Сырьё III: a_III_A = 3 кг на деталь A, a_III_B = 12 кг на деталь B, доступно S_III = 252 кг
Прибыль: p_A = 30 ден. ед. за деталь A, p_B = 40 ден. ед. за деталь B

=== Целевая функция ===
Максимизировать суммарную прибыль:
Z = 30*x_A + 40*x_B -> max

=== Ограничения ===
1) По сырью:
   12*x_A + 4*x_B <= 300 (сырьё I)
   4*x_A + 4*x_B <= 120 (сырьё II)
   3*x_A + 12*x_B <= 252 (сырьё III)
2) Минимальное количество деталей B относительно A: x_B >= x_A
3) Неотрицательность переменных: x_A >= 0, x_B >= 0
```

3) Составить целевую функцию и ограничения - **Z=30xA+40xB→max**

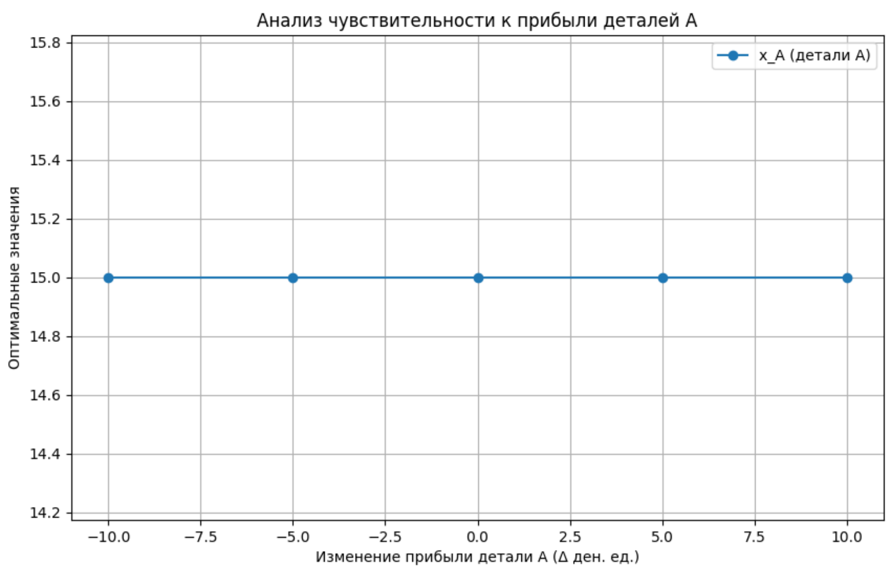
4) Применить графический метод для решения задачи -



5) Провести анализ чувствительность к исходным данным.

```
=== Анализ чувствительности для сырья 1 ===

Изменение сырья: -20 кг -> x_A=15.00, x_B=15.00, Z=1050.00

Изменение сырья: +0 кг -> x_A=15.00, x_B=15.00, Z=1050.00

Изменение сырья: +20 кг -> x_A=15.00, x_B=15.00, Z=1050.00
```

Как видно на рисунке ниже модель не чувствительна к данным



Вывод: были изучены визуальные (графические) методы решения задачь линейного программирования.

Листинг программ

main.py

```
import numpy as np
import matplotlib.pyplot as plt
import math
from time import time


def f(x):
    """Целевая функция: sin²(x) - √x"""
    return np.sin(x) ** 2 - np.sqrt(x)


def find_interval(f, a, b, step=0.01,
max_iter=1000):
    """
    Алгоритм поиска интервала,
содержащего минимум функции
    Возвращает интервал [a, b] такой, что
f(a) > f(c) и f(b) > f(c) для некоторого
c ∈ (a,b)
    """
    x0 = a
    f0 = f(x0)
    x1 = a + step
    f1 = f(x1)

    # Если функция возрастает, идем в
обратном направлении
    if f1 > f0:
        step = -step
        x1 = a + step
        f1 = f(x1)

    # Поиск интервала
    for i in range(max_iter):
        x2 = x1 + step
        f2 = f(x2)

        if f2 > f1:  # Нашли интервал
            if step > 0:
                return (x0, x2)
            else:
                return (x2, x0)
```

```
        # Увеличиваем шаг
        step *= 2
        x0, f0 = x1, f1
        x1, f1 = x2, f2

    return (a, b)  # Если не нашли
подходящий интервал


def dichotomy_method(f, a, b, epsilon,
max_iter=1000):
    """
    Метод дихотомии для поиска минимума
функции
    """
    func_calls = 0
    delta = epsilon / 3  # Малое смещение
для сравнения значений

    for i in range(max_iter):
        if abs(b - a) < epsilon:
            break

        mid = (a + b) / 2
        x1 = mid - delta
        x2 = mid + delta

        f1 = f(x1);
        func_calls += 1
        f2 = f(x2);
        func_calls += 1

        if f1 < f2:
            b = x2
        else:
            a = x1

    x_min = (a + b) / 2
    return x_min, f(x_min), func_calls


def golden_section_method(f, a, b,
epsilon, max_iter=1000):
    """
    Метод золотого сечения для поиска
минимума функции
    """
    func_calls = 0
    phi = (1 + math.sqrt(5)) / 2  #
Золотое сечение
    resphi = 2 - phi
```

```
    x1 = a + resphi * (b - a)
    x2 = b - resphi * (b - a)

    f1 = f(x1);
    func_calls += 1
    f2 = f(x2);
    func_calls += 1

    for i in range(max_iter):
        if abs(b - a) < epsilon:
            break

        if f1 < f2:
            b = x2
            x2 = x1
            f2 = f1
            x1 = a + resphi * (b - a)
            f1 = f(x1);
            func_calls += 1
        else:
            a = x1
            x1 = x2
            f1 = f2
            x2 = b - resphi * (b - a)
            f2 = f(x2);
            func_calls += 1

    x_min = (a + b) / 2
    return x_min, f(x_min), func_calls


def fibonacci_method(f, a, b, epsilon,
max_iter=1000):
    """
    Метод Фибоначчи для поиска минимума
функции
    """
    func_calls = 0

    # Генерируем числа Фибоначчи
    fib = [1, 1]
    while fib[-1] < (b - a) / epsilon:
        fib.append(fib[-1] + fib[-2])

    n = len(fib) - 1

    x1 = a + (fib[n - 2] / fib[n]) * (b -
a)
    x2 = a + (fib[n - 1] / fib[n]) * (b -
a)

    f1 = f(x1);
```

```
    func_calls += 1
    f2 = f(x2);
    func_calls += 1

    for k in range(1, n):
        if f1 > f2:
            a = x1
            x1 = x2
            f1 = f2
            x2 = a + (fib[n - k - 1] /
fib[n - k]) * (b - a)
            if k != n - 1:
                f2 = f(x2);
                func_calls += 1
        else:
            b = x2
            x2 = x1
            f2 = f1
            x1 = a + (fib[n - k - 2] /
fib[n - k]) * (b - a)
            if k != n - 1:
                f1 = f(x1);
                func_calls += 1

    x_min = (a + b) / 2
    return x_min, f(x_min), func_calls


# Параметры задачи
a, b = 0, 1
epsilon_values = [1e-1, 1e-2, 1e-3, 1e-4,
1e-5, 1e-6, 1e-7, 1e-8]

# Поиск интервала, содержащего минимум
interval = find_interval(f, a, b)
print(f"Найденный интервал, содержащий
минимум: {interval}")

# Сравнение методов
results = {
    'Дихотомия': {'calls': [], 'time':
[]},
    'Золотое сечение': {'calls': [],
'time': []},
    'Фибоначчи': {'calls': [], 'time':
[]}
}

for epsilon in epsilon_values:
    print(f"\nТочность ε = {epsilon}")

    # Метод дихотомии
```

```
    start_time = time()
    x_min_d, f_min_d, calls_d =
dichotomy_method(f, interval[0],
interval[1], epsilon)
    time_d = time() - start_time
    results['Дихотомия']
['calls'].append(calls_d)
    results['Дихотомия']
['time'].append(time_d)

    # Метод золотого сечения
    start_time = time()
    x_min_gs, f_min_gs, calls_gs =
golden_section_method(f, interval[0],
interval[1], epsilon)
    time_gs = time() - start_time
    results['Золотое сечение']
['calls'].append(calls_gs)
    results['Золотое сечение']
['time'].append(time_gs)

    # Метод Фибоначчи
    start_time = time()
    x_min_fib, f_min_fib, calls_fib =
fibonacci_method(f, interval[0],
interval[1], epsilon)
    time_fib = time() - start_time
    results['Фибоначчи']
['calls'].append(calls_fib)
    results['Фибоначчи']
['time'].append(time_fib)

    print(f"Дихотомия: x_min =
{x_min_d:.8f}, f_min = {f_min_d:.8f},
вызовов = {calls_d}")
    print(f"Золотое сечение: x_min =
{x_min_gs:.8f}, f_min = {f_min_gs:.8f},
вызовов = {calls_gs}")
    print(f"Фибоначчи: x_min =
{x_min_fib:.8f}, f_min = {f_min_fib:.8f},
вызовов = {calls_fib}")

# Построение графиков
plt.figure(figsize=(15, 10))

# График 1: Количество вычислений функции
от логарифма точности
plt.subplot(2, 2, 1)
log_epsilon = np.log10(epsilon_values)
for method in results:
    plt.plot(log_epsilon, results[method]
['calls'], 'o-', label=method,
```

```
markersize=6)

plt.xlabel('log₁₀(ε)')
plt.ylabel('Количество вычислений
функции')
plt.title('Зависимость количества
вычислений от точности')
plt.grid(True, alpha=0.3)
plt.legend()

# График 2: Время выполнения от логарифма
точности
plt.subplot(2, 2, 2)
for method in results:
    plt.plot(log_epsilon, results[method]
['time'], 's-', label=method,
markersize=6)

plt.xlabel('log₁₀(ε)')
plt.ylabel('Время выполнения (секунды)')
plt.title('Зависимость времени выполнения
от точности')
plt.grid(True, alpha=0.3)
plt.legend()

# График 3: Исходная функция
plt.subplot(2, 2, 3)
x_vals = np.linspace(a, b, 1000)
y_vals = f(x_vals)
plt.plot(x_vals, y_vals, 'b-',
linewidth=2)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Функция f(x) = sin²(x) - √x')
plt.grid(True, alpha=0.3)

# График 4: Отношение количества
вычислений
plt.subplot(2, 2, 4)
dichotomy_calls =
np.array(results['Дихотомия']['calls'])
for method in ['Золотое сечение',
'Фибоначчи']:
    ratio = np.array(results[method]
['calls']) / dichotomy_calls
    plt.plot(log_epsilon, ratio, '^-',
label=f'{method}/Дихотомия',
markersize=6)

plt.xlabel('log₁₀(ε)')
plt.ylabel('Отношение количества
вычислений')
```

```
plt.title('Относительная эффективность
методов')
plt.grid(True, alpha=0.3)
plt.legend()

plt.tight_layout()
plt.show()

# Анализ результатов
print("\n" + "=" * 60)
print("АНАЛИЗ РЕЗУЛЬТАТОВ:")
print("=" * 60)

for i, epsilon in
enumerate(epsilon_values):
    print(f"\nПри ε = {epsilon}:")
    for method in results:
        print(f"{method:15}:
{results[method]['calls'][i]:3d} вызовов,
{results[method]['time'][i]:.6f} сек")


# Поиск максимума (минимум от -f(x))
def negative_f(x):
    return -f(x)


# Находим интервал для максимума
max_interval = find_interval(negative_f,
a, b)
print(f"\nИнтервал, содержащий максимум:
{max_interval}")

# Используем золотое сечение для поиска
максимума
x_max, f_max_neg, calls_max =
golden_section_method(negative_f,
max_interval[0], max_interval[1], 1e-6)
f_max = -f_max_neg

print(f"\nМаксимум функции: x_max =
{x_max:.8f}, f_max = {f_max:.8f}")
print(f"Минимум функции: x_min =
{x_min_gs:.8f}, f_min = {f_min_gs:.8f}")
```