Математические модели систем и их реализация

Содержание

Введение

В общем случае математическую модель некоторой системы можно формализовать в виде некоторого оператора T, преобразующего вектор входных параметров $x \in X$, управляющих воздействий (параметров) $u \in U$, случайных воздействий $\delta \in \Omega$ и времени $t \in \mathbb{R}_+$ в вектор пространства выходных параметров $y \in Y$ (рис. 1). В зависимости от природы конкретной системы математическая форма оператора T может представлять собой уравнение или систему уравнений: алгебраических, трасцендентных, дифференциальных (обыкновенных или в частных производных), интегральных и интегро-дифференциальных.

В зависимости от наличия или отсутствия параметра времени в математическом описании можно выделить статические (время в формулировке задачи отсутствует) и динамические модели (время присутствует в описании). Модели могут учитывать или не учитывать наличие случайных воздействий δ на систему. При отсутствии случайных воздействий в математическом

$$y = T(x, u, \delta, t)$$

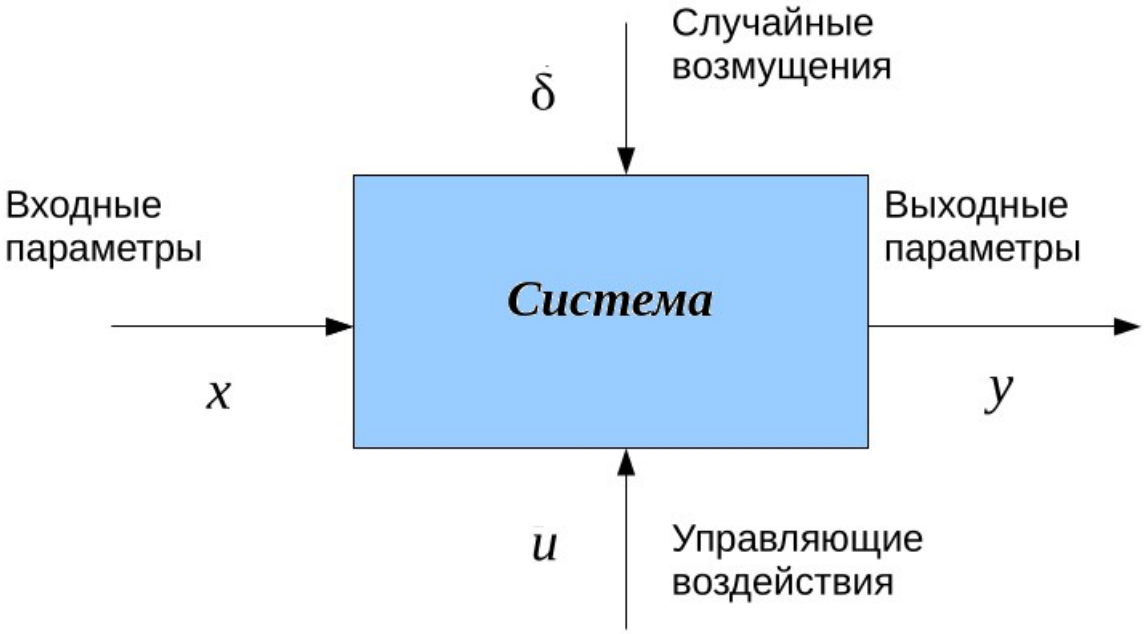


Рисунок 1 - Формализация математической модели

1. Детерминированные модели

В детерминированных моделях каждому значению вектора входных параметров соответствует строго определенное значение вектора выходных. Они основаны на на уравнениях механики, термодинамики, химической кинетики. В этом разделе рассмотрены примеры различных видов детерминированных моделей.

1.1. Статические детерминированные модели

В наиболее простом случае такую математическую модель можно представить в виде системы линейных алгебраических уравнений (СЛАУ)

$$y = A\,x, \tag{1}$$

где A — матрица коэффициентов, содержащая параметры модели.

Часто задача моделирования состоит в том, чтобы по заданному значению вектора выходных параметров y найти отвечающее ему значение входных x . В зависимости от свойств

матрицы A данная задача может быть решена либо непосредственным обращением матрицы (матрица квадратная, невырожденная)

$$x = A^{-1} y, \tag{2}$$

либо одним из вычислительных методов линейной алгебры.

Рассмотрим более подробно основные теоремы и методы решения СЛАУ

$$A x = b, \tag{3}$$

где A - матрица коэффициентов, x — вектор неизвестных, b — вектор свободных членов.

1. Теорема Крамера (условия единственности решения)

- Система n линейных уравнений с n неизвестными имеет единственное решение тогда и только тогда, когда определитель матрицы коэффициентов не равен нулю ($det\ A \neq 0$).

- Если определитель равен нулю, система либо несовместна (нет решений), либо имеет бесконечно много решений (совместна, но не определённа однозначно).

2. Теорема о существовании решения (Кронекера — Капелли)

- Система совместна (имеет хотя бы одно решение), если ранг матрицы коэффициентов равен рангу расширенной матрицы (с учётом свободных членов): rank(A) = rank(A|b)

- Если ранг меньше числа переменных, решений бесконечно много; если меньше — решений нет.

Критерий единственности решения

Если система совместна и:

- rank(A) = n (число переменных) → решение единственно

- rank(A) < n → бесконечное множество решений

3. Теорема о структуре решений

- Любое решение совместной линейной системы представимо как сумма частного решения и произвольной линейной комбинации решений однородной системы.

4. Теорема Фредгольма

Однородная система $Ax = 0$ имеет нетривиальное решение тогда и только тогда, когда $det\left(A\right) = 0$

Основные методы решения систем линейных уравнений

Прямые методы (точные для точной арифметики, сходятся за конечное число итераций)

1. Метод Гаусса

Приведение к ступенчатому виду элементарными преобразованиями.

```
import numpy as np
```

```
A = np.array([[3, 2, -1], [2, -2, 4], [-1, 0.5, -1]])
b = np.array([1, -2, 0])
x = np.linalg.solve(A, b)
```

2. LU-разложение

Приведение матрицы к виду $A=LU$, где L – нижняя треугольная, U – верхняя треугольная матрицы.

```
from scipy.linalg import lu_factor, lu_solve
lu, piv = lu_factor(A)
x = lu_solve((lu, piv), b)
```

3. Метод Холецкого

Для симметричных положительно определенных матриц: $A=LL^T$.

```
L = np.linalg.cholesky(A)
y = np.linalg.solve(L, b)
x = np.linalg.solve(L.T, y)
```

Итерационные методы (для больших разреженных систем)

1. Метод Якоби

$x^{k+1}=D^{-1}\left(b-\left(A-D\right)x^k\right)$, где D – диагональ A.

```
from scipy.sparse.linalg import LinearOperator, gmres
def jacobi_iter(A, b, max_iter=1000):
      D_inv = np.diag(1/np.diag(A))
      x = np.zeros_like(b)
      for _ in range(max_iter):
      x_new = D_inv @ (b - (A - np.diag(np.diag(A))) @ x)
      if np.linalg.norm(x_new - x) < 1e-6:
            break
      x = x_new
      return x
```

2. Метод Гаусса-Зейделя

Использует обновленные значения переменных в текущей итерации.

3. Метод сопряженных градиентов (CG)

Для симметричных положительно определенных матриц.

```
from scipy.sparse.linalg import cg
x, info = cg(A, b, tol=1e-6)
```

4. GMRES

Для несимметричных матриц.

```
from scipy.sparse.linalg import gmres
x, info = gmres(A, b, tol=1e-6, maxiter=1000)
```

В таблице 1 приведены методы и функции numpy, scipy рекомендуемые для решения СЛАУ в зависимости от свойств матрицы коэффициентов.

Таблица 1. Рекомендуемые методы решения СЛАУ

Свойства матрицы	Рекомендуемый метод	Реализация в Python
Плотная, общего вида	LU-разложение	scipy.linalg.solve
Симметричная, положительная	Холецкий / CG	np.linalg.cholesky scipy.sparse.linalg.cg
Ленточная	Специализированные решатели	scipy.linalg.solve_banded
Треугольная	Прямая подстановка	scipy.linalg.solve_triangular
Разреженная общего вида	Итерационные методы (GMRES)	scipy.sparse.linalg.gmres
Разреженная, симметричная	Итерационные методы (CG, MINRES)	scipy.sparse.linalg.cg/minres
Очень большая	Метод Крылова (GMRES, BiCGSTAB)	scipy.sparse.linalg.gmres/ bicgstab

Особые случаи

Вырожденные матрицы ($det\left(A\right)\approx 0$):

```
# Использовать псевдообратную матрицу
x = np.linalg.lstsq(A, b, rcond=None)[0]
```

Переопределенные системы ($m>n$):

```
# Метод наименьших квадратов
x = np.linalg.lstsq(A, b, rcond=None)[0]
```

Недопределенные системы ($m<n$):

```
# Нахождение минимальной нормы
x = np.linalg.pinv(A) @ b
```

Критерии остановки для итерационных методов

1. Норма невязки: $\left\|Ax^k-b\right\|<\varepsilon$

2. Относительное изменение: $\left\|x^k-x^{k-1}\right\|<\varepsilon$

3. Максимальное число итераций

Анализ сходимости

- Необходимое условие: диагональное преобладание

- Достаточное условие: спектральный радиус $\rho\left(I-M^{-1}A\right)<1$

- Ускорение сходимости: Предобуславливание

```
from scipy.sparse.linalg import spilu, LinearOperator
M = spilu(A) # Неполное LU-разложение
M_x = lambda x: M.solve(x)
M_op = LinearOperator(A.shape, M_x)
x, info = gmres(A, b, M=M_op)
```

Практические рекомендации

1. Для небольших матриц (<1000×1000) использовать прямые методы

2. Для разреженных матриц – итерационные методы с предобуславливанием

3. Проверять обусловленность матрицы:

```
cond = np.linalg.cond(A) # Чем больше, тем хуже
```

4. Для симметричных матриц использовать специализированные методы

5. При работе с плохо обусловленными системами:

- Использовать SVD-разложение

- Применять регуляризацию

```
x = np.linalg.pinv(A) @ b # Сингулярное разложение
```

1.2. Примеры реализации статических детерминированных моделей, сводящихся к решению системы линейных уравнений

Статическая определимая стержневая механическая система.

Условие равновесия пространственной конструкции для статически определимых задач формулируется следующим образом:

- Сумма проекций всех сил на каждую ось декартовой системы координат равна нулю

- Сумма моментов всех сил относительно любых осей (обычно координатных) равна нулю

В векторной форме это

$$\begin{aligned} \sum \vec{F}_i = 0 \\ \sum \vec{M}_i = 0 \end{aligned} \tag{4}$$

Аналитическая запись для трёхмерной (3D) системы

$$\begin{aligned} \sum F_x = 0 \\ \sum F_y = 0 \\ \sum F_z = 0 \\ \sum M_x = 0 \\ \sum M_y = 0 \\ \sum M_z = 0 \end{aligned} \tag{5}$$

Для конструкции из брусьев известны направления (через косинусы направлений или углы с осями), силы неизвестны и ищутся как неизвестные составляющие (обычно — продольные усилия в стержнях).

Пусть:
- n — количество элементов (стержней) в конструкции;
- F_k — усилие в k -м элементе;
- $\alpha_k, \beta_k, \gamma_k$ — углы направления оси k -го элемента с осями x, y, z ; или, соответственно, l_k, m_k, n_k — косинусы направлений;
- $\vec{R}_j$ — известные внешние силы (или равнодействующая);
- Конструкция задана топологически (известны точки приложения или узлы).

Каждому узлу соответствует по три уравнения равновесия (по осям).
Примерная запись системы для одной точки (узла)

$$\begin{aligned} \sum_{k \in K} F_k l_k + \sum_j R_{jx} = 0 \\ \sum_{k \in K} F_k m_k + \sum_j R_{jy} = 0, \\ \sum_{k \in K} F_k n_k + \sum_j R_{jz} = 0 \end{aligned} \tag{6}$$

где суммы по k идут по всем элементам, сходящимся в узле; l_k, m_k, n_k — косинусы направления элемента.

Схема Python-программы для вычисления усилий в элементах.

Модель ниже предполагает:

- Для каждого стержня конструктивная информация содержит индексы начального и конечного узлов, а также косинусы направлений (или углы, отразимые в косинусы).
- Внешние силы заданы на узлах.
- Количество уравнений совпадает с числом неизвестных (статически определимая система).

```
import numpy as np

# Данные примера:
# nodes: массив, где каждому узлу соответствует три координаты
nodes = np.array([
[0, 0, 0], # узел 0
[1, 0, 0], # узел 1
# ... (и т.д.)
])

# elements: список (начальный_узел, конечный_узел, угол_x, угол_y, угол_z)
elements = [
(0, 1, alpha1, beta1, gamma1),
# ... далее для всех элементов
]

# Силы: массив размером (N_узлов, 3) – проекции внешней силы на x,y,z в каждом
узле
forces = np.zeros((len(nodes), 3))
forces[0] = [Fx0, Fy0, Fz0] # внешние силы на узел 0
#...

n_equations = 3 * len(nodes) # по три уравнения на узел
n_elements = len(elements)

# Заполнение матрицы коэффициентов
A = np.zeros((n_equations, n_elements))
B = np.zeros(n_equations)

for en, (start, end, alpha, beta, gamma) in enumerate(elements):
l = np.cos(np.radians(alpha))
m = np.cos(np.radians(beta))
n = np.cos(np.radians(gamma))
# Contribution to start node (сила "выходит")
A[3*start, en] += l
A[3*start+1, en] += m
A[3*start+2, en] += n
# Contribution to end node (сила "входит")
```

```
A[3*end, en] -= l
A[3*end+1, en] -= m
A[3*end+2, en] -= n

# Свободный член — отрицательные внешние силы (перенести в левую часть)
B = -forces.flatten()

# Решаем систему
F = np.linalg.solve(A, B)
print('Усилия в элементах:', F)
```

- α, β, γ — углы между осью элемента и осями x, y, z, соответственно (по необходимости заменить на косинусы направлений, если заданы иначе).

Направляющие косинусы определяются для каждого стержня на основании координат начального (i) и конечного (j) узла

$$\vec{d}_{ij} = (x_j - x_i,\ y_j - y_i,\ z_j - z_i) \tag{7}$$

$$|\vec{d}_{ij}| = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \tag{8}$$

$$(l, m, n) = \left(\frac{x_j - x_i}{|\vec{d}_{ij}|},\ \frac{y_j - y_i}{|\vec{d}_{ij}|},\ \frac{z_j - z_i}{|\vec{d}_{ij}|} \right) \tag{9}$$

Программа масштабируемая: достаточно задать правильно массивы nodes, elements, forces, и решение работает для произвольной статически определимой стержневой пространственной конструкции.

Краткие пояснения:

- Если уравнений больше, чем неизвестных (статически неопределимой системы), использовать только уравнения равновесия для расчёта невозможно, нужны дополнительные уравнения типа "совместимость деформаций".
- Если известна только равнодействующая системы, а не все внешние силы, она вводится как единственная внешняя сила в систему.

Ниже приведен рабочий пример программы с моделью пространственной фермы – тетраэдра, удовлетворяющей всем условиям статической определимости:

- 4 узла
- 6 стержней
- Число уравнений (12) = числу неизвестных (6 внутренних + 6 реакций)

```
"""
ПРОГРАММА ДЛЯ РАСЧЕТА ПРОСТРАНСТВЕННЫХ ФЕРМ
"""

import numpy as np
```

```
def solve_spatial_truss():
    """
    Решение статически определимой пространственной фермы
    с заданными условиями закрепления
    """

    print("=== РАСЧЕТ СТАТИЧЕСКИ ОПРЕДЕЛИМОЙ ПРОСТРАНСТВЕННОЙ ФЕРМЫ ===")
    print()

    # Пространственная ферма: расширенный тетраэдр
    # 4 узла, 6 стержней + правильные опоры

    # Координаты узлов
    nodes = np.array([
        [0, 0, 0],    # 0 - основание (полностью закреплен)
        [3, 0, 0],    # 1 - свободный узел
        [0, 3, 0],    # 2 - свободный узел
        [0, 0, 3],    # 3 - нагруженный узел
    ])

    # Стержни (все пары узлов для жесткости)
    elements = [
        (0, 1),  # основание
        (0, 2),  # основание
        (1, 2),  # основание
        (0, 3),  # к верхнему узлу
        (1, 3),  # к верхнему узлу
        (2, 3),  # к верхнему узлу
    ]

    # КЛЮЧЕВАЯ ОСОБЕННОСТЬ: правильные опорные условия
    # Для 3D системы нужно исключить 6 степеней свободы жесткого тела
    # Используем схему: шарнирно-неподвижная + две шарнирно-подвижных опоры

    restraints = [
        (0, [1, 1, 1]),  # узел 0: x, y, z (шарнирно-неподвижная) - 3 реакции
        (1, [1, 1, 0]),  # узел 1: x, y (скользящая по z) - 2 реакции
        (2, [0, 0, 1]),  # узел 2: z (скользящая по x,y) - 1 реакция
    ]
    # Итого: 6 реакций - ровно столько, сколько нужно для устранения движений жесткого тела

    # Внешние силы только на свободный узел 3
    forces = np.zeros((4, 3))
    forces[1] = [5, 0, 0]      # горизонтальная сила
    forces[2] = [0, -8, 0]     # вертикальная сила
    forces[3] = [0, 0, -12]    # сила по оси z
```

```
print("Геометрия фермы:")
for i, coord in enumerate(nodes):
    print(f"Узел {i}: ({coord[0]}, {coord[1]}, {coord[2]})")

print("\nСтержни:")
for i, (start, end) in enumerate(elements):
    d = nodes[end] - nodes[start]
    L = np.linalg.norm(d)
    print(f"Стержень {i}: узел {start}-{end}, L={L:.2f}м")

print("\nОпорные условия:")
directions = ['x', 'y', 'z']
for node_idx, mask in restraints:
    blocked = [directions[i] for i in range(3) if mask[i]]
    print(f"Узел {node_idx}: заблокировано {', '.join(blocked)}")

print("\nНагрузки:")
for i in range(1,len(forces)):
    print(f" Сила {i}: Fx={forces[i][0]:.0f} кН, Fy={forces[i][1]:.0f} кН, Fz={forces[i][2]:.0f} кН")

# Подсчет размерности системы
n_nodes = len(nodes)
n_elems = len(elements)
n_react = sum(sum(mask) for _, mask in restraints)
n_unknowns = n_elems + n_react
n_equations = n_nodes * 3

print(f"\n=== АНАЛИЗ РАЗМЕРНОСТИ ===")
print(f"Узлов: {n_nodes}")
print(f"Стержней: {n_elems}")
print(f"Опорных реакций: {n_react}")
print(f"Всего неизвестных: {n_unknowns}")
print(f"Уравнений равновесия: {n_equations}")

# Проверка условия статической определимости
print(f"\nПроверка статической определимости:")
print(f"Условие для 3D ферм: m = 3j - 6")
print(f"3j - 6 = 3×{n_nodes} - 6 = {3*n_nodes - 6}")
print(f"m (стержней) = {n_elems}")

if n_elems == 3*n_nodes - 6:
    print("✓ Условие для стержневой части выполнено")

print(f"Общий баланс: {n_unknowns} неизвестных, {n_equations} уравнений")

# Построение матрицы системы уравнений
print(f"\n=== СОСТАВЛЕНИЕ СИСТЕМЫ УРАВНЕНИЙ ===")
```

```
    A = np.zeros((n_equations, n_unknowns))
    b = -forces.flatten()  # внешние силы с противоположным знаком

    # Заполнение коэффициентов для стержней
    print("Направляющие косинусы стержней:")
    for num, (i, j) in enumerate(elements):
        d = nodes[j] - nodes[i]
        L = np.linalg.norm(d)
        l, m, n = d / L  # направляющие косинусы

        print(f"  {num}: l={l:.3f}, m={m:.3f}, n={n:.3f}")

        # Влияние на узел i (положительное направление усилия от узла)
        A[3*i:3*i+3, num] += [l, m, n]

        # Влияние на узел j (отрицательное направление)
        A[3*j:3*j+3, num] -= [l, m, n]

    # Заполнение коэффициентов для опорных реакций
    print("\nОпорные реакции:")
    reaction_idx = n_elems
    for node_idx, mask in restraints:
        for axis in range(3):
            if mask[axis]:
                row = 3 * node_idx + axis
                A[row, reaction_idx] = 1.0
                print(f"  R{reaction_idx-n_elems+1}: узел {node_idx}, направление
{directions[axis]}")
                reaction_idx += 1

    print(f"\nМатрица системы: {A.shape}")
    print(f"Ранг матрицы: {np.linalg.matrix_rank(A)}")

    # Решение системы
    print(f"\n=== РЕШЕНИЕ СИСТЕМЫ ===")

    if A.shape[0] == A.shape[1]:
        # Квадратная система
        det = np.linalg.det(A)
        print(f"Определитель: {det:.2e}")

        if abs(det) > 1e-12:
            print("✓ Система невырожденная - решаем точно")
            solution = np.linalg.solve(A, b)
        else:
            print("⚠️ Система вырожденная - используем псевдообращение")
            solution = np.linalg.pinv(A) @ b

    elif A.shape[0] > A.shape[1]:
```

```
    # Переопределенная система
    print("Система переопределена - используем метод наименьших квадратов")
    solution, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)

    print(f"Ранг: {rank}")
    if len(residuals) > 0:
        print(f"Сумма квадратов остатков: {residuals[0]:.2e}")

else:
    # Недоопределенная система
    print("Система недоопределена - используем псевдообращение")
    solution = np.linalg.pinv(A) @ b

# Извлечение результатов
member_forces = solution[:n_elems]
reactions = solution[n_elems:]

# Вывод результатов
print(f"\n=== РЕЗУЛЬТАТЫ РАСЧЕТА ===")

print("\nУсилия в стержнях:")
print("(+) растяжение, (-) сжатие")
for i, force in enumerate(member_forces):
    start, end = elements[i]
    state = "растяжение" if force > 0 else "сжатие" if force < 0 else "нулевое"
    print(f"  Стержень {i} ({start}-{end}): {force:8.2f} кН ({state})")

print("\nОпорные реакции:")
reaction_idx = 0
for node_idx, mask in restraints:
    for axis in range(3):
        if mask[axis]:
            print(f"  Узел {node_idx}, {directions[axis]}: {reactions[reaction_idx]:8.2f} кН")
            reaction_idx += 1

# Проверка решения
print(f"\n=== ПРОВЕРКА РЕШЕНИЯ ===")
residual = A @ solution - b
max_error = np.max(np.abs(residual))

print(f"Максимальная невязка: {max_error:.2e}")

if max_error < 1e-10:
    print("✓ Решение точное")
elif max_error < 1e-6:
    print("✓ Решение с хорошей точностью")
else:
    print("⚠️ Возможны ошибки в решении")
```

```
    # Проверка равновесия узлов
    print("\nПроверка равновесия узлов:")
    for node in range(n_nodes):
        node_forces = np.zeros(3)

        # Внешние силы
        node_forces += forces[node]

        # Внутренние силы от стержней
        for elem_idx, (i, j) in enumerate(elements):
            if i == node:
                d = nodes[j] - nodes[i]
                L = np.linalg.norm(d)
                direction = d / L
                node_forces += member_forces[elem_idx] * direction
            elif j == node:
                d = nodes[j] - nodes[i]
                L = np.linalg.norm(d)
                direction = d / L
                node_forces -= member_forces[elem_idx] * direction

        # Реакции опор
        reaction_idx = 0
        for support_node, mask in restraints:
            if support_node == node:
                for axis in range(3):
                    if mask[axis]:
                        node_forces[axis] += reactions[reaction_idx]
                        reaction_idx += 1
            else:
                reaction_idx += sum(mask)

        equilibrium_error = np.linalg.norm(node_forces)
        status = "✓" if equilibrium_error < 1e-6 else "⚠"
        print(f"  Узел {node}: {status} {equilibrium_error:.2e}")

    return {
        'nodes': nodes,
        'elements': elements,
        'restraints': restraints,
        'forces': forces,
        'member_forces': member_forces,
        'reactions': reactions,
        'matrix_A': A,
        'solution': solution
    }

if __name__ == "__main__":
    # Выполнение расчета
```

```
results = solve_spatial_truss()

print(f"\n" + "="*60)
print("ПРОГРАММА ЗАВЕРШЕНА УСПЕШНО")
print("="*60)
```

Модель межотраслевого баланса Леонтьева (экономика).

Одна из самых известных детерминированных моделей, приводящих к системе линейных уравнений — это межотраслевой баланс В.А. Леонтьева.

Суть модели

- Экономика разбивается на n отраслей.
- Каждая отрасль потребляет продукцию других отраслей и производит свой конечный продукт.
- Для определения объёмов производства x_i каждой отрасли составляется следующая система:

$$x_i = \sum_{j=1}^{n} a_{ij} x_j + y_i, \tag{10}$$

где

- x_i — валовый выпуск i-й отрасли,
- a_{ij} — коэффициент затрат продукции j-й отрасли для производства единицы продукции i-й отрасли,
- y_i — конечное потребление продукции i-й отрасли.

В матричном виде

$$x = Ax + y \tag{11}$$

$$(I - A)x = y \tag{12}$$

где I — единичная матрица, A — матрица прямых затрат.

Требуется найти x ; для этого решается система линейных уравнений.
Пример Python-кода
Рассмотрим экономику из 3 отраслей.

```
import numpy as np

# Матрица прямых затрат (A): a_ij — сколько продукции i-й отрасли нужно для производства 1 ед. продукции j-й отрасли
A = np.array([
    [0.2, 0.1, 0.0],
    [0.05, 0.2, 0.05],
    [0.0, 0.05, 0.1]
])

# Вектор конечного спроса (спрос на продукцию отраслей)
y = np.array([100, 80, 40])
```

```
# Единичная матрица
I = np.eye(A.shape[0])

# Решение: x = (I - A)^(-1) * y
L = np.linalg.inv(I - A)   # Леонтьевская обратная матрица
x = L @ y

print("Валовый выпуск по отраслям:")
for i, val in enumerate(x, 1):
    print(f"Отрасль {i}: {val:.2f}")
```

Результат расчета:

Валовый выпуск по отраслям:

Отрасль 1: 138.98

Отрасль 2: 111.85

Отрасль 3: 50.66

Метод материального баланса в химической технологии.

В химии расчет составов потоков в стационарной технологической схеме также приводит к системе линейных уравнений.

Пример: материальный баланс для реактора и разделительных аппаратов

- Рассматриваются входящие, выходящие и промежуточные потоки веществ.

- Для каждого элемента (и соединения) составляется баланс:

$$\sum_j a_{ij} x_j = b_i \qquad (13)$$

где

- x_j — расход потока j,

- a_{ij} — коэффициент (стехиометрия, участие вещества в потоке),

- b_i — внешний приток или отток вещества i.

Типовой пример — схема с двумя реакторами и смесителем:

- Неизвестны массовые расходы всех потоков.

- Для каждого вещества составляется линейное уравнение баланса.

- Система легко записывается в матричной форме и решается стандартными методами линейной алгебры.

Пример: Материальный баланс химической установки

Рассмотрим классический инженерный пример — расчет стационарного материального баланса для схемы с несколькими потоками.

Условие задачи

На химическом производстве имеются три сырьевых потока (Feed1, Feed2, Feed3) с известным массовым составом по компонентам A, B и C. Два потока (Feed1, Feed2) заданы по массе (в кг/ч) и составу, третий поток (Feed3) имеет известный состав, но его расход неизвестен. Все три сырьевых потока подаются в установку, где происходит перераспределение компонентов на два продукта:

- Product1, содержащий только компоненты A и B в массовых долях 50:50.

- Product2, состоящий из 90% C и 10% B.

Необходимо определить:

- Массу дополнительного сырьевого потока Feed3.

- Массу выходных продуктов Product1 и Product2.

Задача требует соблюдения материального баланса по каждому компоненту (A, B, C), а также общего масс-баланса: суммарная масса сырья = суммарная масса продуктов.

Математическая модель

Обозначения:

- $F_1 = 100$ кг/ч — расход Feed1

- $F_2 = 150$ кг/ч — расход Feed2

- F_3 — расход Feed3 (неизвестен)

- x_1 — масса Product1 (неизвестна)

- x_2 — масса Product2 (неизвестна)

Массовые доли:

| Поток | A | B | C |

| Feed1 | 0.60 | 0.35 | 0.05 |

| Feed2 | 0.00 | 0.85 | 0.15 |

| Feed3 | 0.20 | 0.30 | 0.50 |

| Product1 | 0.50 | 0.50 | 0.00 |

| Product2 | 0.00 | 0.10 | 0.90 |

Модель основана на балансе компонентов:

1. Баланс по компоненту A:

$$0.60 \cdot F_1 + 0.00 \cdot F_2 + 0.20 \cdot F_3 = 0.50 \cdot x_1$$

2. Баланс по компоненту B:

$$0.35 \cdot F_1 + 0.85 \cdot F_2 + 0.30 \cdot F_3 = 0.50 \cdot x_1 + 0.10 \cdot x_2$$

3. Баланс по компоненту C:

$$0.05 \cdot F_1 + 0.15 \cdot F_2 + 0.50 \cdot F_3 = 0.90 \cdot x_2$$

Переносим все выражения в одну сторону, чтобы получить каноническую форму системы:

$$\begin{cases} 0.20 \cdot F_3 - 0.50 \cdot x_1 = -(0.60 \cdot F_1) \\ 0.30 \cdot F_3 - 0.50 \cdot x_1 - 0.10 \cdot x_2 = -(0.35 \cdot F_1 + 0.85 \cdot F_2) \\ 0.50 \cdot F_3 - 0.90 \cdot x_2 = -(0.05 \cdot F_1 + 0.15 \cdot F_2) \end{cases}$$

Эта система представлена в матричной форме:

$$A \cdot \begin{bmatrix} F_3 \\ x_1 \\ x_2 \end{bmatrix} = b,$$

где:

$$A = \begin{bmatrix} 0.2 & -0.5 & 0.0 \\ 0.3 & -0.5 & -0.1 \\ 0.5 & 0.0 & -0.9 \end{bmatrix}, \quad b = \begin{bmatrix} -0.6 \cdot F_1 \\ -0.35 \cdot F_1 - 0.85 \cdot F_2 \\ -0.05 \cdot F_1 - 0.15 \cdot F_2 \end{bmatrix}.$$

Решение задачи

Модель сводится к решению системы линейных уравнений с тремя неизвестными: F_3, x_1, x_2. После расчёта также выполняется проверка общего масс-баланса:

$$F_1 + F_2 + F_3 = x_1 + x_2$$

Если равенство соблюдается с заданной точностью, баланс считается корректным.

Пример кода для решения

```
import numpy as np

# Исходные потоки
F1 = 100  # Feed1, кг/ч
F2 = 150  # Feed2, кг/ч

# Feed1: 60%A, 35%B, 5%C
# Feed2: 0%A, 85%B, 15%C
# Feed3: 20%A, 30%B, 50%C — F3 неизвестен
```

```
# Product1: 50%A, 50%B
# Product2: 10%B, 90%C

# Неизвестны переменные: F3, x1, x2

# Формируем систему уравнений по компонентным балансам: A, B, C
# входы - выходы = 0

A = np.array([
    [0.2, -0.5,   0.0],    # A: 0.2·F3 - 0.5·x1 = - (0.6·F1 + 0.0·F2)
    [0.3, -0.5,  -0.1],    # B: 0.3·F3 - 0.5·x1 - 0.1·x2 = - (0.35·F1 + 0.85·F2)
    [0.5,  0.0,  -0.9],    # C: 0.5·F3 - 0.9·x2 = - (0.05·F1 + 0.15·F2)
])

b = np.array([
    - (0.6 * F1 + 0.0 * F2),
    - (0.35 * F1 + 0.85 * F2),
    - (0.05 * F1 + 0.15 * F2),
])

# Решение
sol = np.linalg.solve(A, b)
F3, x1, x2 = sol

# Вывод результатов
print(f"Feed3:     {F3:.2f} кг/ч")
print(f"Product1:  {x1:.2f} кг/ч")
print(f"Product2:  {x2:.2f} кг/ч")

# Проверка общего масс-баланса
total_in = F1 + F2 + F3
total_out = x1 + x2

print(f"\nСуммарный вход:  {total_in:.2f} кг/ч")
print(f"Суммарный выход: {total_out:.2f} кг/ч")

if np.isclose(total_in, total_out, rtol=1e-6):
    print("✅ Масс-баланс соблюдён")
else:
    print("❌ Нарушение масс-баланса!")
```

В результате выполнения кода посчитаются массовые расходы всех потоков на основе полной системы материального баланса:

Feed3: -2237.50 кг/ч

Product1: -775.00 кг/ч

Product2: -1212.50 кг/ч

Суммарный вход: -1987.50 кг/ч

Суммарный выход: -1987.50 кг/ч

✅ Масс-баланс соблюдён

Комментарии

- В реальных задачах число потоков и веществ может быть существенно больше, и система уравнений становится размерностью $n \times n$, где n — число неизвестных.

- Подобные системы при проектировании химических технологических схем решают пакетным способом с помощью линейной алгебры (LU, QR, SVD).

- Этот подход с развернутой системой уравнений — основа автоматизированных расчетных программ для хим. индустрии.

1.3. Примеры статических детерминированных моделей, описываемых уравнениями в частных производных

В настоящем разделе рассмотрены примеры решения двумерных задач для математических моделей, описываемых линейными уравнениями в частных производных. Такие модели описывают равновесные состояния физических систем (упругих тел, электростатических полей, стационарных потоков). Математический аппарат таких моделей основан на линейных дифференциальных операторах в частных производных, действующих на функции из $\mathbb{R}^2$.

Математическая суть модели.

Линейный оператор:

Модель задается уравнением вида

$$L[u](x,y) = f(x,y), \quad (x,y) \in \Omega \subset \mathbb{R}^2, \tag{14}$$

где:

$u(x,y)$ — искомая функция (напряжения, потенциал, перемещения),

L — линейный дифференциальный оператор (эллиптического типа),

f — заданная функция (внешние нагрузки, источники).

Примеры операторов:

Лапласиан: $L = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ (электростатика, теплопроводность),

Бигармонический оператор: $L = \nabla^4 = \nabla^2(\nabla^2)$ (линейная теория упругости),

Оператор Гельмгольца: $L = \nabla^2 + k^2$ (акустика).

Суть постановки задачи.

Краевая задача:

Требуется найти функцию u, удовлетворяющую:

Уравнению $L[u]=f$ в области Ω,

Граничным условиям на $\partial\Omega$:

Дирихле: $u|_{\partial\Omega}=g$,

Неймана: $\left.\frac{\partial u}{\partial n}\right|_{\partial\Omega}=h$,

Смешанным: $\alpha u+\beta\left.\frac{\partial u}{\partial n}\right|_{\partial\Omega}=\gamma$.

Физическая интерпретация для теории упругости:

u — функция напряжений - функция Эри (Ф),

$L=\nabla^4$ — бигармонический оператор,

Граничные условия выражают баланс поверхностных сил.

Методы решения.

Аналитические подходы:

- Метод функции Эри: Сведение задачи к бигармоническому уравнению $\nabla^4\Phi=0$ с последующим подбором решений (полиномы, ряды).
- Разделение переменных: Поиск решений в виде $u(x,y)=X(x)Y(y)$.
- Интегральные преобразования: Использование преобразований Фурье/Лапласа.

Вариационные методы:

Формулировка задачи минимизации:

Решение u минимизирует функционал энергии:

$$J[u]=\frac{1}{2}\int_{\Omega}\left(\left|\nabla^2 u\right|^2\right)dxdy-\int_{\Omega} f\,u\,dxdy\quad(\text{для}\ \nabla^4 u=f). \tag{15}$$

Метод Ритца:

Аппроксимация u комбинацией базисных функций: $u_N = \sum_{k=1}^{N} c_k \phi_k$, сведение к системе линейных уравнений для c_k.

Конечные элементы: Дискретизация области и построение решения на сетке.

Универсальность подхода:

Вариационная формулировка единообразно применима к широкому классу операторов (эллиптических, бигармонических), связывая физические законы с принципами минимума энергии.

1.3.1. Уравнение Лапласа и методы его решения

Уравнение Лапласа — одно из фундаментальных уравнений математической физики. Оно описывает стационарные (не зависящие от времени) процессы в отсутствие источников или стоков поля. Это эллиптическое уравнение в частных производных (УрЧП) второго порядка.

Стандартная форма (в трехмерном декартовом пространстве):

$$\Delta u = 0 \text{ или } \nabla^2 u = 0, \tag{16}$$

где:

- $u = u(x, y, z)$ — искомая функция (например, потенциал, температура, давление).
- $\Delta = \nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2 + \partial^2/\partial z^2$ — оператор Лапласа.

Физический смысл: Уравнение Лапласа описывает состояние равновесия поля:

- Электростатика: Потенциал φ в области без зарядов ($\rho = 0$).
- Теплопроводность: Стационарное распределение температуры T в теле без источников тепла.
- Гидродинамика: Потенциал скорости φ для несжимаемой ($\nabla \cdot v = 0$) и безвихревой ($\nabla \times v = 0$) жидкости.
- Гравитация: Гравитационный потенциал φ в области без масс.
- Теория упругости: Функции напряжений.

Краевые условия: Так как уравнение Лапласа описывает состояние внутри области, для его решения необходимо задать условия на границе S области V. Основные типы:

1. Задача Дирихле: На границе заданы значения искомой функции.

   $u|_S = g(s)$

   Пример: Температура на поверхности тела поддерживается постоянной.

2. Задача Неймана: На границе задана нормальная производная искомой функции.

$$\left.\frac{\partial u}{\partial n}\right|_S = h(s)$$

Пример: Задан тепловой поток через поверхность тела.

3. Смешанная задача: На разных частях границы заданы условия Дирихле и Неймана.

Основные методы решения

Решение уравнения Лапласа сильно зависит от геометрии области и типа краевых условий. Методы делятся на две большие группы:

I. Аналитические (Точные) Методы:

Применимы для областей простой геометрии (прямоугольник, круг, сфера, цилиндр).

1. Метод разделения переменных:
   - Суть: Предполагают, что решение можно представить в виде произведения функций, каждая из которых зависит только от одной координаты: $u(x,y,z) = X(x)Y(y)Z(z)$.
   - Процесс: Подстановка в уравнение Лапласа приводит к разделению на обыкновенные дифференциальные уравнения (ОДУ) для $X(x), Y(y), Z(z)$. Решают эти ОДУ, получая семейства решений. Удовлетворяют краевым условиям, подбирая константы разделения и составляя решение в виде ряда (часто ряда Фурье).
   - Применение: Прямоугольные области (декартовы координаты), круговые области (полярные/цилиндрические координаты), сферические области (сферические координаты).
2. Метод функций комплексного переменного (для 2D задач):
   - Суть: Использует мощный аппарат теории функций комплексного переменного (ТФКП). Решение $u(x,y)$ ищется как вещественная (или мнимая) часть аналитической функции $f(z) = u(x,y) + i\,v(x,y), (z = x + iy)$.
   - Конформные отображения: Позволяет решать задачу для сложной области, отображая ее конформно на более простую область (например, круг или полуплоскость), где решение известно или легко находится.
   - Интегральные формулы: Формула Шварца, Формула Пуассона для круга и полуплоскости дают явное решение задачи Дирихле для этих областей.
   - Применение: Двумерные задачи электростатики, гидродинамики, теории упругости.

3. Метод интегральных преобразований:
    - Суть: Применение преобразований Фурье, Лапласа, Ханкеля к уравнению Лапласа по одной или нескольким переменным. Это преобразует УрЧП в ОДУ (для преобразованной функции), которое решается легче. Затем применяют обратное преобразование.
    - Применение: Задачи с бесконечными или полубесконечными областями (например, полупространство), задачи с осевой симметрией (преобразование Ханкеля).
4. Метод функции Грина:
    - Суть: Решение представляется в виде интеграла от функции Грина $G(r, r_0)$ по границе области. Функция Грина представляет собой потенциал точечного источника (заряда) в данной области с заданными граничными условиями (часто однородными).
    - Применение: Элегантное формальное решение для задач Дирихле и Неймана. Позволяет свести решение УрЧП к решению более простого уравнения для G и последующему интегрированию. Точное вычисление G возможно только для простых областей.

II. Численные Методы:

Применяются для областей сложной геометрии, где аналитическое решение недоступно.

1. Метод конечных разностей (МКР / FDM):
    - Суть: Область решения покрывается дискретной сеткой. Производные в операторе Лапласа аппроксимируются разностными отношениями на этой сетке. Уравнение Лапласа превращается в систему линейных алгебраических уравнений (СЛАУ) относительно значений функции u в узлах сетки.
    - Типы сеток: Равномерные, неравномерные, структурированные.
    - Решение СЛАУ: Прямые методы (Гаусса, LU-разложение - для небольших сеток) или итерационные методы (Якоби, Гаусса-Зейделя, Либмана, последовательной верхней релаксации - SOR - для больших сеток).
    - Достоинства: Простота реализации для прямоугольных областей.
    - Недостатки: Сложность аппроксимации сложных границ, необходимость мелкой сетки для точности.
2. Метод конечных элементов (МКЭ / FEM):
    - Суть: Область разбивается на множество небольших взаимосвязанных подобластей - конечных элементов (треугольники, четырехугольники в 2D; тетраэдры, гексаэдры в 3D). Искомая функция u аппроксимируется простыми

функциями (часто полиномами) внутри каждого элемента. Идея метода Ритца-Галеркина применяется к вариационной постановке задачи Дирихле для уравнения Лапласа (минимизация функционала Дирихле). Это приводит к большой, но разреженной СЛАУ.

- Достоинства: Гибкость в работе со сложными геометриями и границами, естественная адаптация к локальным особенностям решения.
- Недостатки: Более сложная математическая основа и реализация по сравнению с МКР.

3. Метод граничных элементов (МГЭ / BEM):

- Суть: Основан на интегральных представлениях решения через потенциалы (например, с использованием функции Грина). Дискретизации подвергается только граница области (а не вся область, как в МКР и МКЭ). Уравнение Лапласа сводится к интегральному уравнению на границе, которое затем дискретизируется, приводя к СЛАУ относительно значений u или $\partial u/\partial n$ на границе.
- Достоинства: Уменьшение размерности задачи (только граница), высокая точность при расчете граничных значений и производных, хорошо подходит для задач в бесконечных областях.
- Недостатки: Матрица системы СЛАУ - плотная (не разреженная), сложность при неоднородных средах, сложная реализация для нелинейных задач.

Ключевые выводы:

1. Универсальность: Уравнение Лапласа описывает равновесные состояния в самых разных физических полях без источников.

2. Граничные условия: Определяют тип задачи (Дирихле, Нейман, смешанная) и являются неотъемлемой частью постановки.

3. Выбор метода: Зависит от:

- Геометрии области.
- Типа краевых условий.
- Требуемой точности и детализации решения.
- Доступных вычислительных ресурсов.

4. Аналитика *vs* Численные методы: Аналитические методы дают точное решение в замкнутой форме, но применимы только к простым областям. Численные методы универсальны для сложных областей, но дают приближенное решение и требуют вычислительных ресурсов.

Пример аналитического решения уравнения Лапласа в двумерном случае с использованием метода разделения переменных

Задача: Найти стационарное распределение температуры $u(x,y)$ внутри прямоугольной пластины ($0 \le x \le a, 0 \le y \le b$). Уравнение Лапласа в двумерной задаче имеет вид

$$\nabla^2 u = 0, \left(\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 = 0\right). \quad (17)$$

Границы пластины поддерживаются при следующих температурах:

$$u(0,y) = 0, \text{ (левая грань)}, \quad (18)$$

$$u(a,y) = 0, \text{ (правая грань)}, \quad (19)$$

$$u(x,0) = 0, \text{ (нижняя грань)}, \quad (20)$$

$$u(x,b) = f(x), \text{ (верхняя грань - заданная функция)}. \quad (21)$$

Решение методом разделения переменных:

1. Предположение: Будем искать решение в виде произведения функций, каждая из которых зависит только от одной переменной:

$$u(x,y) = X(x) \cdot Y(y)$$

2. Подстановка в уравнение Лапласа:

Подставим $u(x,y) = X(x)Y(y)$ в $\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 = 0$:

$$Y(y)\frac{d^2X}{dx^2} + X(x)\frac{d^2Y}{dy^2} = 0$$

3. Разделение переменных:

Поделим обе части уравнения на $X(x)Y(y)$(предполагая $X(x) \neq 0$ и $Y(y) \neq 0$):

$$\frac{1}{X(x)}\frac{d^2X}{dx^2} + \frac{1}{Y(y)}\frac{d^2Y}{dy^2} = 0$$

Перегруппируем:

$$\frac{1}{X(x)}\frac{d^2X}{dx^2} = -\frac{1}{Y(y)}\frac{d^2Y}{dy^2}$$

Левая часть зависит только от x, правая - только от y. Это возможно только если обе части равны одной и той же константе. Обозначим эту константу через $-\lambda$ (знак выбран для удобства дальнейшего удовлетворения граничным условиям):

$$\frac{1}{X(x)}\frac{d^2X}{dx^2} = -\lambda$$

$$-\frac{1}{Y(y)}\frac{d^2Y}{dy^2}=-\lambda$$

Получаем два обыкновенных дифференциальных уравнения (ОДУ):

$$\frac{d^2X}{dx^2}+\lambda X=0, \tag{22}$$

$$\frac{d^2Y}{dy^2}-\lambda Y=0. \tag{23}$$

4. Удовлетворение однородным граничным условиям:

Применим граничные условия по x и y, которые равны нулю:

$u(0,y)=X(0)Y(y)=0\Rightarrow X(0)=0$ (для всех y)

$u(a,y)=X(a)Y(y)=0\Rightarrow X(a)=0$ (для всех y)

$u(x,0)=X(x)Y(0)=0\Rightarrow Y(0)=0$ (для всех x)

Решаем уравнение (22) для X(x) с граничными условиями X(0)=0, X(a)=0:

Уравнение $X''+\lambda X=0$ - классическое уравнение на собственные значения.

- Случай $\lambda<0$: Решение $X(x)=C_1e^{\sqrt{|\lambda|}x}+C_2e^{-\sqrt{|\lambda|}x}$. Условия $X(0)=0,\ X(a)=0$ приводят только к тривиальному решению $C_1=C_2=0$ (не подходит).

- Случай $\lambda=0$: Решение $X(x)=C_1x+C_2$. Условия $X(0)=0\Rightarrow C_2=0$;

  $X(a)=0\Rightarrow C_1a=0\Rightarrow C_1=0$. Снова тривиальное решение.

- Случай $\lambda>0$: Решение $X(x)=C_1\cos(\sqrt{\lambda}x)+C_2\sin(\sqrt{\lambda}x)$.

  - $X(0)=0: C_1\cdot1+C_2\cdot0=0\Rightarrow C_1=0.$
  - $X(a)=0: C_2\sin(\sqrt{\lambda}a)=0.$

  Чтобы избежать тривиального решения ($C_2=0$), нужно:

$$\sin(\sqrt{\lambda}a)=0\Rightarrow\sqrt{\lambda}a=n\pi\Rightarrow\sqrt{\lambda}=n\pi/a\Rightarrow\lambda=\lambda_n=(n\pi/a)^2,$$

где $n=1,2,3,\ldots.$

Собственные функции: $X_n(x)=\sin((n\pi x)/a)$

5. Решаем уравнение (23) для $Y(y)$ с граничным условием $Y(0)=0$ и найденными λ_n:

Уравнение (23) с $\lambda=\lambda_n=(n\pi/a)^2$

$$\frac{d^2Y}{dy^2} - ((n\pi)/a)^2 Y = 0 \quad (24)$$

Это линейное однородное ОДУ с постоянными коэффициентами. Характеристическое уравнение: $r^2 - (n\pi/a)^2 = 0 \Rightarrow r = \pm(n\pi/a)$.

Общее решение: $Y(y) = A_n \cosh((n\pi y)/a) + B_n \sinh((n\pi y)/a)$, где cosh и sinh - гиперболические косинус и синус.

Применяем граничное условие $Y(0) = 0$:

$Y(0) = A_n \cdot 1 + B_n \cdot 0 = 0 \Rightarrow A_n = 0.$

Собственные решения для $Y(y)$: $Y_n(y) = \sinh((n\pi y)/a)$

6. Построение частных решений:

Для каждого n получаем частное решение исходного уравнения Лапласа, удовлетворяющее всем трем однородным граничным условиям:

$$u_n(x, y) = X_n(x) Y_n(y) = \sin((n\pi x)/a) \sinh((n\pi y)/a). \quad (25)$$

7. Суперпозиция:

Так как уравнение Лапласа линейное и однородное, сумма частных решений также будет решением. Общее решение, удовлетворяющее первым трем граничным условиям, имеет вид ряда:

$$u(x, y) = \Sigma_{n=1}^{\infty} C_n \sin((n\pi x)/a) \sinh((n\pi y)/a) \quad (26)$$

где C_n - пока неизвестные коэффициенты.

8. Удовлетворение неоднородному граничному условию (на верхней грани):

Применим четвертое граничное условие $u(x, b) = f(x)$:

$u(x, b) = \Sigma_{n=1}^{\infty} C_n \sin((n\pi x)/a) \sinh((n\pi b)/a) = f(x)$

Обозначим $D_n = C_n \sinh((n\pi b)/a)$. Тогда:

$$\Sigma_{n=1}^{\infty} D_n \sin((n\pi x)/a) = f(x) \quad (27)$$

Уравнение (27) представляет собой разложение Фурье по синусам функции $f(x)$ на интервале $[0, a]$.

9. Определение коэффициентов D_n (и C_n):

Коэффициенты ряда Фурье-синусов находятся по формуле:

$D_n = (2/a) \int_0^a f(x) \sin((n\pi x)/a) \, dx$

Зная D_n, находим C_n:

$$C_n = D_n / \sinh\left((n\pi b)/a\right) = \frac{(2/a)\int_0^a f(x)\sin\left((n\pi x)/a\right)dx}{\sinh\left((n\pi b)/a\right)}$$

10. Окончательное решение:

Подставляя найденные C_n в ряд (26), получаем аналитическое решение задачи:

$$u(x,y) = \sum_{n=1}^{\infty} \frac{\frac{2}{a}\int_0^a f(\xi)\sin\left(\frac{n\pi\xi}{a}\right)d\xi}{\sinh\left(\frac{n\pi b}{a}\right)} \sin\left(\frac{n\pi x}{a}\right)\sinh\left(\frac{n\pi y}{a}\right) \qquad (28)$$

Физическая интерпретация: Это решение описывает установившуюся температуру в любой точке (x, y) прямоугольной пластины. Температура равна нулю на трех гранях и задана функцией f(x) на верхней грани. Ряд Фурье обеспечивает выполнение этого условия наверху, а гиперболический синус sinh обеспечивает плавное изменение температуры от нуля на нижней грани (y=0) до заданного распределения f(x) на верхней грани (y=b).

Ключевые моменты примера:

1. Разделение переменных свело УрЧП к ОДУ.

2. Однородные граничные условия по x привели к задаче Штурма-Лиувилля, породившей собственные значения λ_n и собственные функции $X_n(x)$.

3. Решение для $Y(y)$ с учетом λ_n и граничного условия при $y = 0$.

4. Принцип суперпозиции для линейных уравнений позволил построить общее решение в виде ряда.

5. Неоднородное граничное условие при $y = b$ определило коэффициенты Фурье C_n разложения функции $f(x)$ по собственным функциям $\sin\left((n\pi x)/a\right)$.

Этот пример демонстрирует методику применения аналитических методов для областей простой геометрии. Для конкретной функции $f(x)$ (например, $f(x) = U_0 = const$ или $f(x) = U_0 \sin(\pi x/a)$) можно вычислить интеграл и получить явный вид решения.

Ниже приведен код Python, решающий данную задачу с использованием пакета sympy.

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Определяем символы с указанием свойств
```

```
x, y, a, b = sp.symbols('x y a b', real=True, positive=True)
n = sp.symbols('n', integer=True, positive=True)
U0 = sp.symbols('U0', real=True)

# Граничная функция
f_x = U0 * sp.sin(sp.pi * x / a)

# Вычисляем коэффициенты Фурье
D_n = (2 / a) * sp.integrate(f_x * sp.sin(n * sp.pi * x / a), (x, 0, a))
D_n = sp.simplify(D_n)

# Упрощаем с учетом целочисленности n
D_n = sp.trigsimp(D_n).rewrite(sp.Piecewise)
D_n = sp.simplify(D_n)

# Вычисляем коэффициенты C_n
C_n = D_n / sp.sinh(n * sp.pi * b / a)

# Формируем решение
u_sol = sp.Sum(C_n * sp.sin(n * sp.pi * x / a) * sp.sinh(n * sp.pi * y / a), (n, 1, sp.oo))

# Аналитически упрощаем решение
solution = sp.simplify(u_sol.doit())

# Создаем компактное представление
if solution.has(sp.Piecewise):
    # Извлекаем ненулевые компоненты
    n1_solution = solution.args[0].args[0][0]  # Решение для n=1
    general_solution = sp.Sum(solution.args[0].args[1][0], (n, 2, sp.oo))
    compact_solution = n1_solution + general_solution
else:
    compact_solution = solution

# Генерируем LaTeX
latex_output = sp.latex(
    compact_solution,
    mode='plain',
    fold_frac_powers=False,
    long_frac_ratio=2,
    mul_symbol='dot',
    ln_notation=True
)

print("Аналитическое решение уравнения Лапласа:\n")
print(latex_output)

# Альтернативный вывод через pprint

sp.pprint(compact_solution, use_unicode=True)
```

```
# Визуализация 3D

# Параметры задачи (можно менять)
a_val = 1.0       # Ширина пластины
b_val = 1.0       # Высота пластины
U0_val = 100.0      # Амплитуда температуры
N_terms = 20      # Количество членов ряда для численного расчета

# Численная реализация для визуализации
def calculate_u(x_val, y_val, N=20):
    """Вычисление решения в точке (x_val, y_val) с N членами ряда"""
    total = 0.0
    for ni in range(1, N + 1):
        # Вычисляем коэффициент C_n для текущего n
        Cn = C_n.subs({
            a: a_val,
            b: b_val,
            U0: U0_val,
            n: ni
        }).evalf()

        # Вычисляем член ряда
        term = Cn * np.sin(ni * np.pi * x_val / a_val) * np.sinh(ni * np.pi * y_val / b_val)
        total += term
    return total

# Создаем сетку для расчета
x_vals = np.linspace(0, a_val, 50)
y_vals = np.linspace(0, b_val, 50)
X, Y = np.meshgrid(x_vals, y_vals)

# Рассчитываем значения температуры на сетке
Z = np.zeros_like(X)
for i in range(len(x_vals)):
    for j in range(len(y_vals)):
        Z[j, i] = calculate_u(x_vals[i], y_vals[j], N_terms)

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Настройка осей перед добавлением данных
ax.set_xlabel('x', fontsize=12, labelpad=10)
ax.set_ylabel('y', fontsize=12, labelpad=10)
ax.set_zlabel('u(x,y)', fontsize=12, labelpad=15, rotation=90)
ax.zaxis.set_rotate_label(False)  # Важная настройка!

# Добавление поверхности
surf = ax.plot_surface(X, Y, Z, cmap='viridis',
```

```
                edgecolor='none', alpha=0.8)

# Настройка заголовка
title = f'Решение уравнения Лапласа\nГраничное условие: u(x,{b_val}) = '
if f_x == U0 * sp.sin(sp.pi * x / a):
    title += r'$U_0 \sin(\pi x / a)$'
else:
    title += str(f_x.subs({a: a_val, U0: U0_val}))
ax.set_title(title, fontsize=14)

# Настройка вида
ax.view_init(elev=30, azim=45)
ax.grid(True)

# Добавляем цветовую шкалу
fig.colorbar(surf, ax=ax, shrink=0.5, aspect=10, label='Температура')

plt.tight_layout()
plt.show()

# Проверка граничных условий (сечение при y=b)
y_top = b_val
u_top = [calculate_u(x_val, y_top, N_terms) for x_val in x_vals]

plt.figure(figsize=(10, 5))
plt.plot(x_vals, u_top, 'b-', linewidth=2, label='Вычисленное решение')
plt.plot(x_vals, [f_x.subs({x: xv, a: a_val, U0: U0_val}).evalf() for xv in x_vals],
         'r--', linewidth=1.5, label='Граничное условие')
plt.title(f'Проверка граничного условия при y = {b_val}', fontsize=14)
plt.xlabel('x', fontsize=12)
plt.ylabel(f'u(x, {b_val})', fontsize=12)
plt.legend()
plt.grid(True)

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

Программа выводит результат вычислений в терминал в формате Latex а также в виде псевдографики с помощью функции pprint:

$$\sum_{n=2}^{\infty} \frac{U_0 \cdot \sin\left(\frac{\pi \cdot n \cdot x}{a}\right) \cdot \sinh\left(\frac{\pi \cdot n \cdot y}{a}\right)}{\sinh\left(\frac{\pi \cdot b \cdot n}{a}\right)}$$

Также производится построение графика распределения температуры (рис. 2) и сравнение результатов аналитического решения уравнения Лапласа с граничными условиями на верхней стороне пластины (рис. 3). Данный график демонстрирует точность аналитического решения данной задачи, поскольку кривая распределения температуры на верхней границе пластины полностью совпадает с заданным граничным условием.

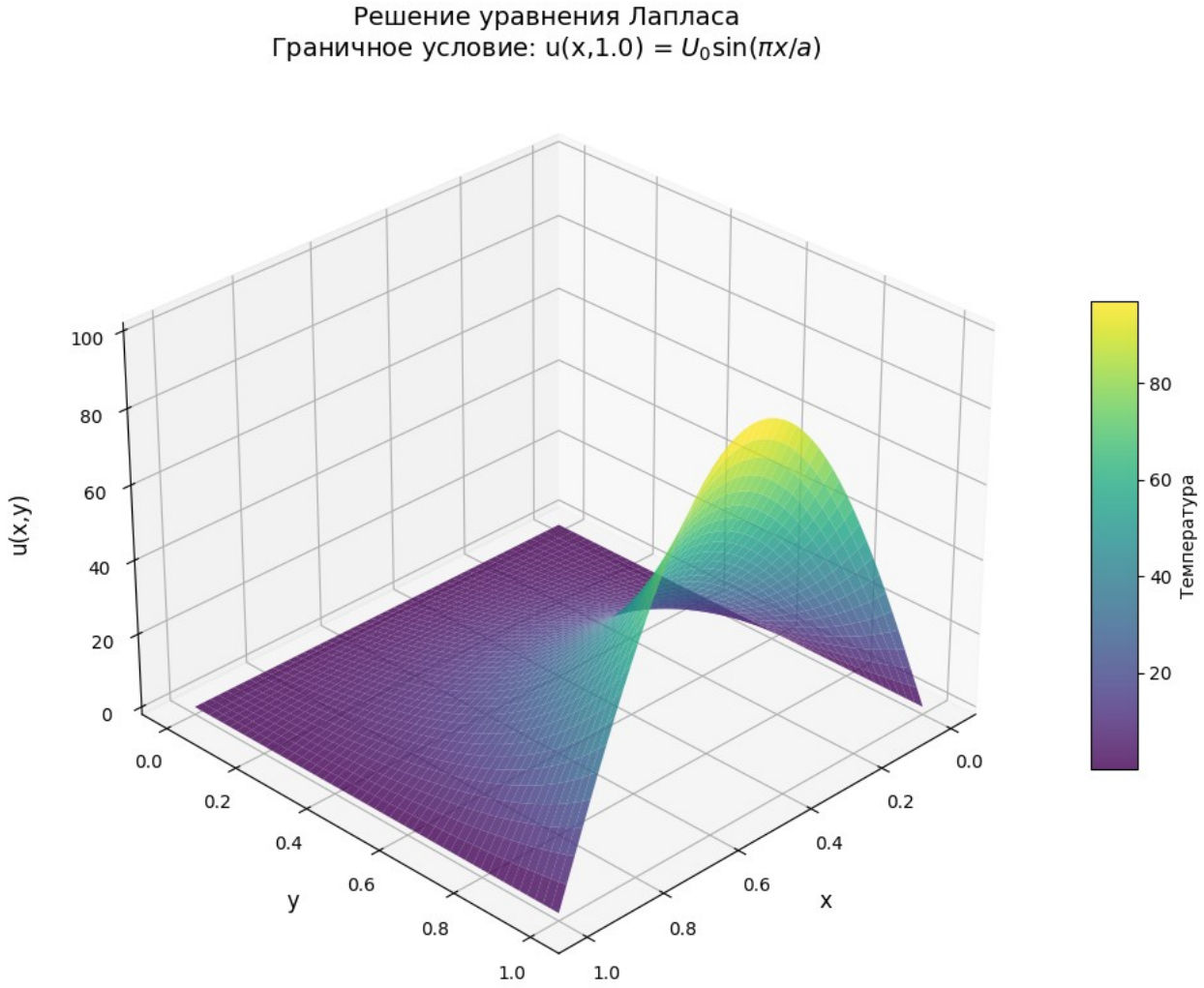


Рисунок 2 - Распределение температуры в пластине по результатам аналитического решения

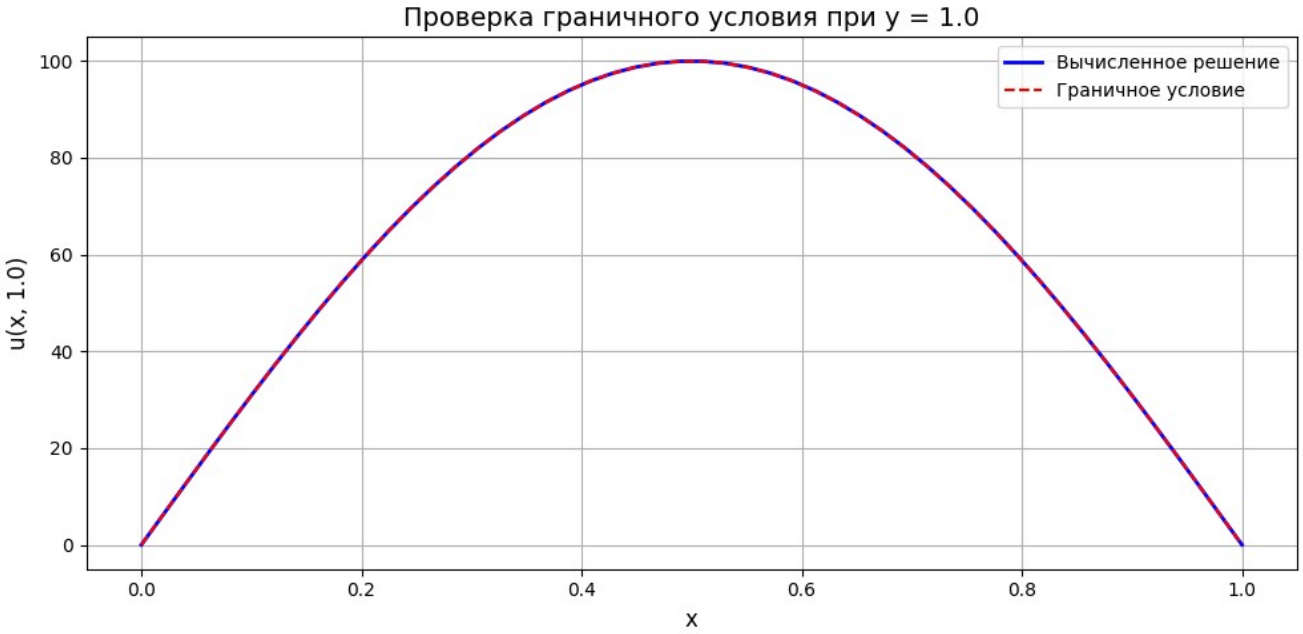


Рисунок 3 - Сравнение результатов аналитического решения уравнения Лапласа с граничным условием на верхней стороне пластины (кривые полностью совпадают).

Численное решение задачи методом конечных разностей

Рассмотрим численный алгоритм решения данной задачи методом конечных разностей: уравнение (17) с граничными условиями (18)-(21). Функция, заданная на границе (21) имеет вид

$$f(x)=U_0\sin(\pi x/a) \tag{29}$$

Шаг 1: Дискретизация области

Разделим прямоугольную область $[0,a]\times[0,b]$ на равномерную сетку:

- N_x узлов по оси x

- N_y узлов по оси y

- Шаг сетки: $h_x=a/(N_x-1), h_y=b/(N_y-1)$

- Узлы: $x_i=i\cdot h_x\,(i=0,1,\ldots,N_x-1),\ y_j=j\cdot h_y\,(j=0,1,\ldots,N_y-1)$

Шаг 2: Аппроксимация производных

Используем центральные разности второго порядка:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} \tag{30}$$

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} \tag{31}$$

Подставляем в уравнение Лапласа:

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} = 0 \tag{32}$$

Шаг 3: Получение итерационной формулы

Выразим u_{ij}:

$$u_{i,j} = \frac{\frac{u_{i-1,j} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} + u_{i,j+1}}{h_y^2}}{\frac{2}{h_x^2} + \frac{2}{h_y^2}} \tag{33}$$

Упростим коэффициенты:

$$\alpha = \frac{h_y^2}{2\left(h_x^2 + h_y^2\right)}$$
$$\beta = \frac{h_x^2}{2\left(h_x^2 + h_y^2\right)} \tag{34}$$

Итерационная формула:

$$u_{i,j}^{(k+1)} = \alpha\left(u_{i-1,j}^{(k)} + u_{i+1,j}^{(k)}\right) + \beta\left(u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)}\right) \tag{35}$$

Шаг 4: Алгоритм решения

```
import numpy as np

# Параметры задачи
a, b = 1.0, 1.0          # Размеры области
U0 = 1.0                 # Амплитуда
```

```
f = lambda x: U0*np.sin(np.pi*x/a)  # Граничное условие

# Параметры сетки
Nx, Ny = 50, 50            # Количество узлов
hx = a/(Nx-1)              # Шаг по x
hy = b/(Ny-1)              # Шаг по y

# Коэффициенты
α = hy2 / (2*(hx2 + hy2))
β = hx2 / (2*(hx2 + hy2))

# Инициализация сетки
U = np.zeros((Ny, Nx))

# Применение граничных условий
x = np.linspace(0, a, Nx)
U[-1, :] = f(x)   # Верхняя граница: u(x, b) = f(x)
U[:, 0] = 0       # Левая граница: u(0, y) = 0
U[:, -1] = 0      # Правая граница: u(a, y) = 0
U[0, :] = 0       # Нижняя граница: u(x, 0) = 0

# Итерационный процесс
max_iter = 10000
tolerance = 1e-5

for k in range(max_iter):
    U_old = U.copy()

    # Обновление внутренних точек (i=1..Nx-2, j=1..Ny-2)
    for j in range(1, Ny-1):
        for i in range(1, Nx-1):
            U[j, i] = α*(U_old[j, i-1] + U_old[j, i+1]) + β*(U_old[j-1, i] + U_old[j+1, i])

    # Проверка сходимости
    max_diff = np.max(np.abs(U - U_old))
    if max_diff < tolerance:
        print(f"Сходимость достигнута на итерации {k+1}")
        break
else:
    print("Сходимость не достигнута")


 Шаг 5: Векторизованная версия (для ускорения)

# ... (инициализация и граничные условия остаются прежними)

for k in range(max_iter):
    U_old = U.copy()
```

```
    # Векторизованное обновление
    U[1:-1, 1:-1] = α*(U_old[1:-1, :-2] + U_old[1:-1, 2:]) + β*(U_old[:-2, 1:-1] + U_old[2:, 1:-1])

    # Проверка сходимости
    max_diff = np.max(np.abs(U - U_old))
    if max_diff < tolerance:
        break
```

Шаг 6: Оптимизация методом SOR

```
ω = 1.8  # Параметр релаксации (1 < ω < 2)

for k in range(max_iter):
    max_diff = 0
    for j in range(1, Ny-1):
        for i in range(1, Nx-1):
            new_val = α*(U[j, i-1] + U[j, i+1]) + β*(U[j-1, i] + U[j+1, i])
            diff = new_val - U[j, i]
            U[j, i] += ω * diff
            if abs(diff) > max_diff:
                max_diff = abs(diff)

    if max_diff < tolerance:
        break
```

Пояснения:

1. Аппроксимация производных:

- Центральные разности обеспечивают точность $O(h^2)$

- Для внутренних точек используется пятиточечный шаблон ("крест")

2. Итерационная схема:

- Метод Якоби: используют значения с предыдущей итерации (U_{old})

- Метод Гаусса-Зейделя: используют уже обновленные значения (ускоряет сходимость)

- SOR (Successive Over-Relaxation): ускорение сходимости с параметром ω

3. Сходимость:

- Итерации продолжаются, пока максимальное изменение между итерациями не станет меньше tolerance

- Теоретическая скорость сходимости $O(h^2)$

4. Граничные условия:

- Фиксируются перед итерациями и не изменяются в процессе

- Для задач Дирихле значения на границе остаются постоянными

Визуализация результата:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Создание сетки
X, Y = np.meshgrid(np.linspace(0, a, Nx), np.linspace(0, b, Ny))

# 3D визуализация
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, U, cmap='viridis', edgecolor='none')

ax.set_title('Численное решение уравнения Лапласа', fontsize=14)
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.set_zlabel('u(x,y)', fontsize=12, rotation=90)
ax.zaxis.set_rotate_label(False)

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

Особенности реализации:

1. Гибкость: Легко модифицировать под другие граничные условия

2. Производительность: Векторизованная версия в 50-100 раз быстрее

3. Точность: Контроль сходимости через параметр tolerance

4. Ускорение: Метод SOR сокращает количество итераций в 5-10 раз

Данный алгоритм обеспечивает эффективное численное решение уравнения Лапласа для прямоугольных областей с точностью, достаточной для большинства практических приложений.

Ниже приводится полный код программы, сравнивающей аналитический и численный метод решения данной задачи.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Параметры задачи
a = 1.0      # Ширина пластины
```

```
b = 1.0      # Высота пластины
U0 = 100.0     # Амплитуда температуры
f = lambda x: U0 * np.sin(np.pi * x / a)  # Граничное условие сверху

# Параметры сетки
Nx = 50      # Узлов по x
Ny = 50      # Узлов по y
hx = a / (Nx - 1)  # Шаг по x
hy = b / (Ny - 1)  # Шаг по y

# Создание сетки
x = np.linspace(0, a, Nx)
y = np.linspace(0, b, Ny)
X, Y = np.meshgrid(x, y)

# Инициализация решения
U_num = np.zeros((Ny, Nx))

# Граничные условия
U_num[-1, :] = f(x)    # Верхняя граница: u(x, b) = f(x)
U_num[:, 0] = 0        # Левая граница: u(0, y) = 0
U_num[:, -1] = 0       # Правая граница: u(a, y) = 0
U_num[0, :] = 0        # Нижняя граница: u(x, 0) = 0

# Коэффициенты для итераций
alpha = hy**2 / (2*(hx**2 + hy**2))
beta = hx**2 / (2*(hx**2 + hy**2))

# Итерационный процесс (метод Либмана)
max_iter = 10000
tolerance = 1e-5
for iteration in range(max_iter):
    U_old = U_num.copy()

    # Обновление внутренних точек (векторизованная версия)
    U_num[1:-1, 1:-1] = alpha * (U_num[1:-1, :-2] + U_num[1:-1, 2:]) + beta * (U_num[:-2, 1:-1]
+ U_num[2:, 1:-1])

    # Проверка сходимости
    delta = np.max(np.abs(U_num - U_old))
    if delta < tolerance:
        print(f'Сходимость достигнута на итерации {iteration+1}, Δ={delta:.2e}')
        break
else:
    print(f'Сходимость не достигнута за {max_iter} итераций, Δ={delta:.2e}')

# Аналитическое решение
def analytical_solution(x, y):
    return U0 * np.sin(np.pi * x / a) * np.sinh(np.pi * y / a) / np.sinh(np.pi * b / a)
```

```
U_analytical = analytical_solution(X, Y)

# Разность решений
diff = U_num - U_analytical

# 1. График с обоими решениями
fig1 = plt.figure(figsize=(12, 8))
ax1 = fig1.add_subplot(111, projection='3d')

# Поверхность численного решения (полупрозрачная)
surf_num = ax1.plot_surface(X, Y, U_num, cmap='viridis',
                            alpha=0.7, label='Численное решение')

# Поверхность аналитического решения (проволочная сетка)
surf_anal = ax1.plot_wireframe(X, Y, U_analytical, color='red',
                               linewidth=0.7, label='Аналитическое решение')

# Настройки графика
ax1.set_title('Сравнение численного и аналитического решений', fontsize=14)
ax1.set_xlabel('x', fontsize=12)
ax1.set_ylabel('y', fontsize=12)
ax1.set_zlabel('u(x,y)', fontsize=12, rotation=90)
ax1.zaxis.set_rotate_label(False)
ax1.legend()
ax1.view_init(elev=30, azim=-45)

# 2. График разности решений
fig2 = plt.figure(figsize=(12, 8))
ax2 = fig2.add_subplot(111, projection='3d')

# Поверхность разности
surf_diff = ax2.plot_surface(X, Y, diff, cmap='coolwarm',
                             edgecolor='none', alpha=0.8)

# Настройки графика
ax2.set_title('Разность решений: численное - аналитическое', fontsize=14)
ax2.set_xlabel('x', fontsize=12)
ax2.set_ylabel('y', fontsize=12)
ax2.set_zlabel('Δu', fontsize=12, rotation=90)
ax2.zaxis.set_rotate_label(False)

# Добавляем цветовую шкалу
cbar = fig2.colorbar(surf_diff, ax=ax2, shrink=0.6, aspect=10)
cbar.set_label('Погрешность', fontsize=12)

# Сечение при y = b/2 для детального сравнения
y_half = Ny // 2
plt.figure(figsize=(10, 6))
```

```
plt.plot(x, U_num[y_half, :], 'b-', linewidth=2, label='Численное')
plt.plot(x, U_analytical[y_half, :], 'r--', linewidth=1.5, label='Аналитическое')
plt.plot(x, diff[y_half, :], 'g:', linewidth=1.5, label='Разность')

plt.title(f'Сравнение решений при y = {y[y_half]:.2f}', fontsize=14)
plt.xlabel('x', fontsize=12)
plt.ylabel('u(x,y)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()

# Оценка погрешности
abs_error = np.max(np.abs(diff))
rel_error = np.max(np.abs(diff)) / np.max(np.abs(U_analytical))
print(f'Максимальная абсолютная погрешность: {abs_error:.4e}')
print(f'Максимальная относительная погрешность: {rel_error:.4e}')
print(f'Среднеквадратичная погрешность: {np.sqrt(np.mean(diff**2)):.4e}')

plt.show()
```

Результаты расчета показывают на хорошее согласие аналитического и численного решений данной задачи:

Сходимость достигнута на итерации 4273, Δ=9.98e-06

Максимальная абсолютная погрешность: 8.2757e-03

Максимальная относительная погрешность: 8.2800e-05

Среднеквадратичная погрешность: 3.4986e-03

Трехмерные графики распределения температуры, полученные аналитическим и численным методами практически неразличимы (рис. 4) . Трехмерный график невязок (рис. 5) показывает незначительную величину абсолютной ошибки численного метода, что подтверждают графики распределения температуры в среднем сечении пластины, полученные аналитическим и численным методами (рис. 6).

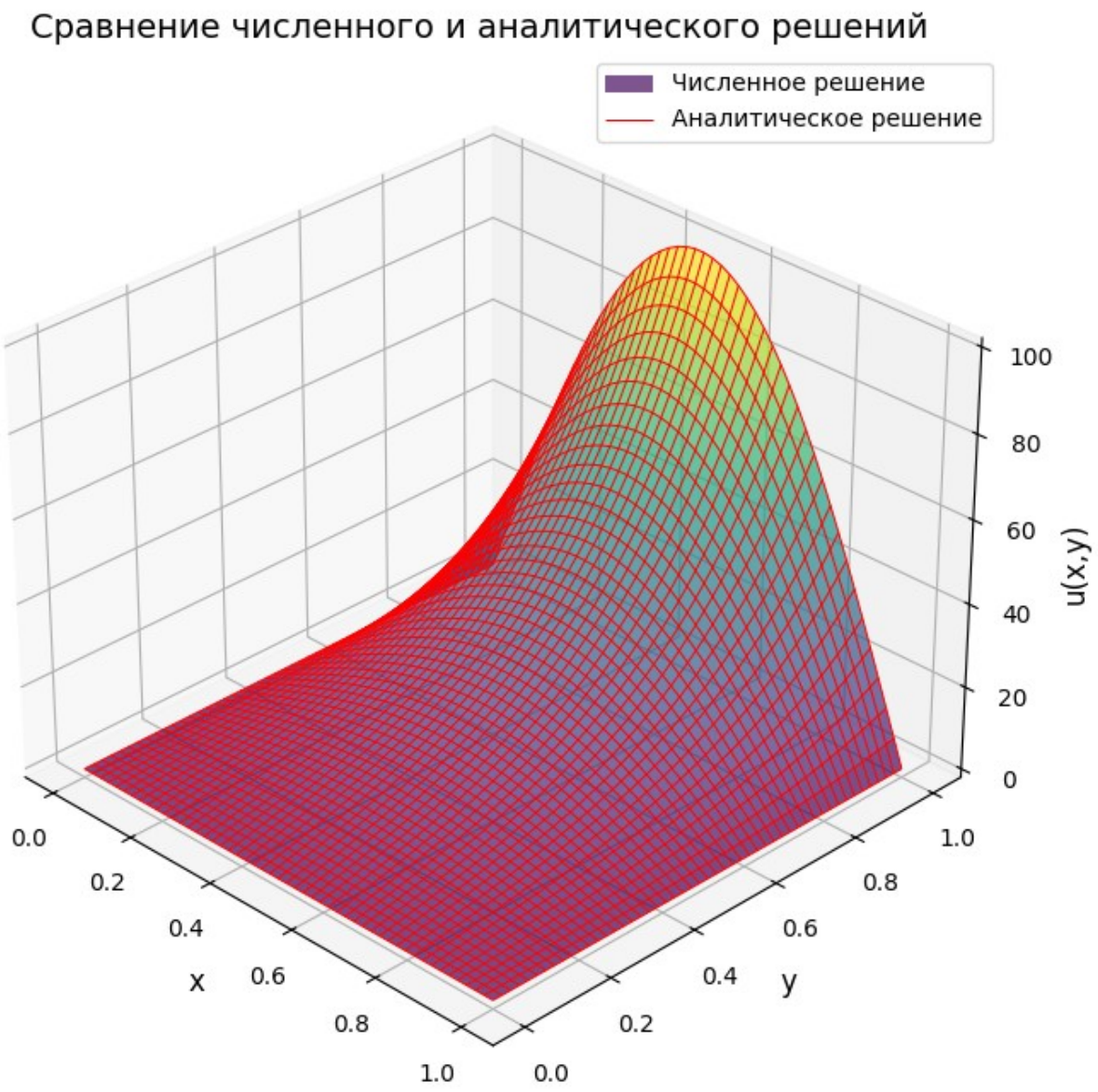


Рисунок 4 - Трехмерные графики аналитического и численного решения уравнения Лапласа в одних и тех же координатах

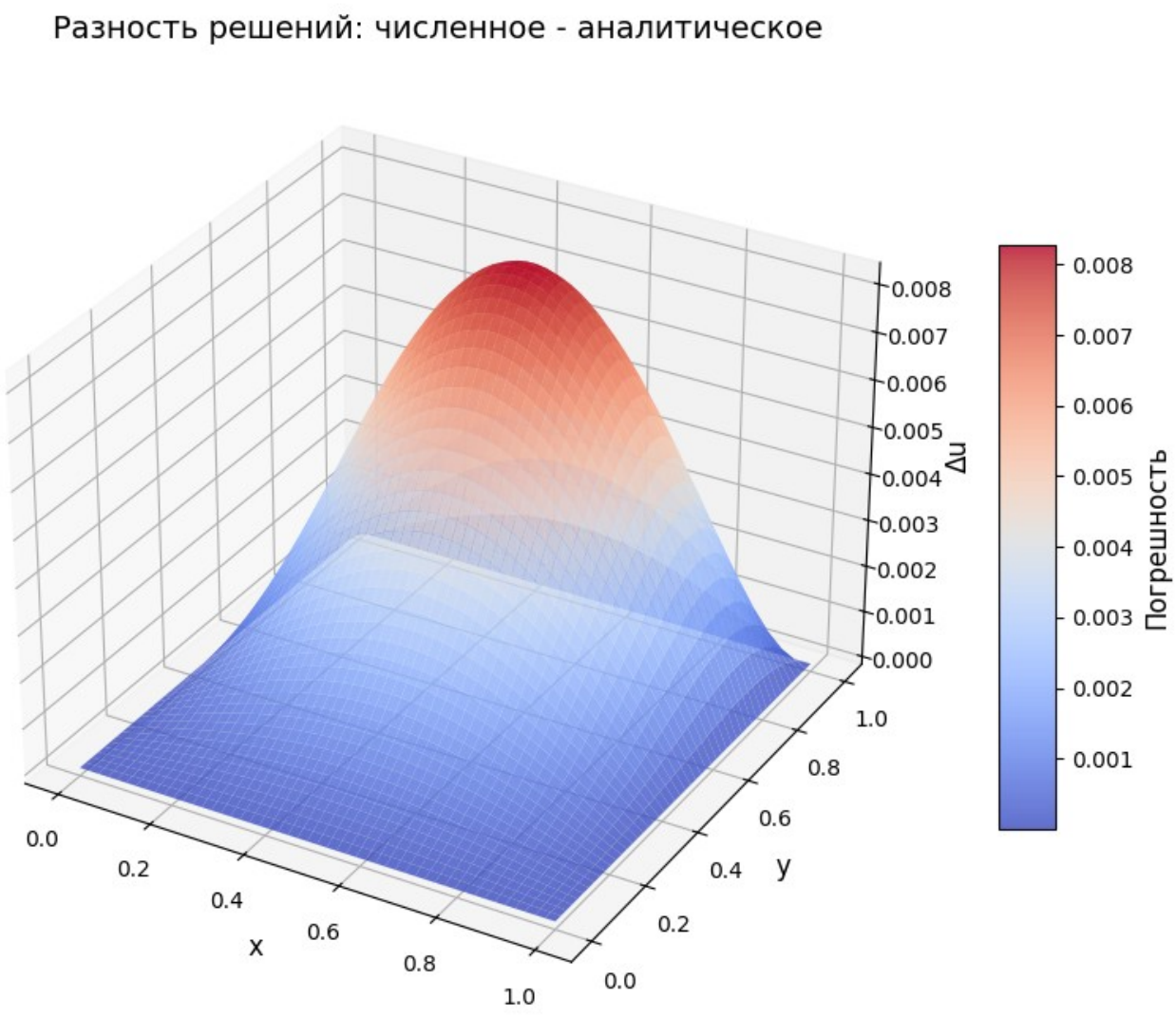


Рисунок 5 - График распределения погрешности численного решения уравнения Лапласа

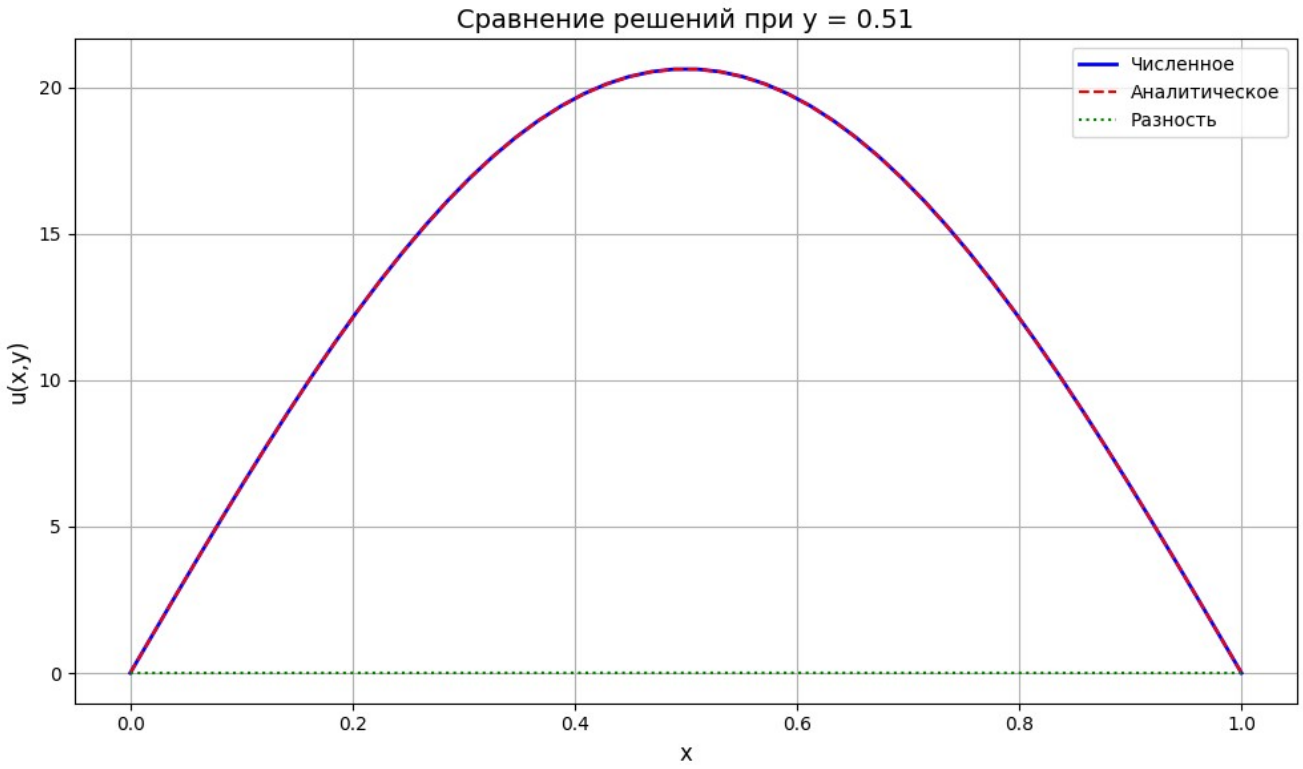


Рисунок 6 - Сравнение аналитического и численного решений в средней части пластины

1.3.2. Основные уравнения плоской задачи теории упругости

1. Тензор напряжений (Stress Tensor)

Физический смысл:

Характеризует внутренние силы, возникающие в деформируемом теле. В двумерном случае это симметричный тензор 2-го ранга

$$\sigma = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{pmatrix} \tag{36}$$

где:

σ_{xx}, σ_{yy} — нормальные напряжения (растяжение/сжатие вдоль осей),

$\sigma_{xy} = \sigma_{yx}$ — касательные напряжения (сдвиговые усилия).

Уравнения равновесия (баланс сил)

$$\begin{cases} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} = 0 \\ \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} = 0 \end{cases} \quad (37)$$

Интерпретация:

Сумма проекций всех сил на оси x и y равна нулю. Это следует из второго закона Ньютона для статики.

2. Тензор деформаций (Strain Tensor)

Физический смысл:

Описывает геометрические изменения формы тела. В линейной теории упругости

$$\varepsilon = \begin{pmatrix} \varepsilon_{xx} \varepsilon_{xy} \\ \varepsilon_{yx} \varepsilon_{yy} \end{pmatrix} \quad (38)$$

где:

ε_{xx}, ε_{yy} — относительные удлинения вдоль осей,

$\varepsilon_{xy} = \varepsilon_{yx} = \frac{1}{2}\gamma_{xy}$ — сдвиговые деформации (γ_{xy} — угол сдвига).

Связь с перемещениями (u_x, u_y — компоненты вектора перемещений)

$$\begin{aligned} \varepsilon_{xx} &= \frac{\partial u_x}{\partial x}, \\ \varepsilon_{yy} &= \frac{\partial u_y}{\partial y}, \\ \varepsilon_{xy} &= \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \end{aligned} \quad (39)$$

3. Уравнение совместности деформаций (Compatibility Equation)

$$\frac{\partial^2 \varepsilon_{xx}}{\partial y^2} + \frac{\partial^2 \varepsilon_{yy}}{\partial x^2} = 2\frac{\partial^2 \varepsilon_{xy}}{\partial x \partial y} \quad (40)$$

Физический смысл:

Гарантирует, что деформации ε_{ij} соответствуют непрерывному полю перемещений $u_i(x, y)$. Без этого условия деформированное тело могло бы иметь "разрывы" или "наложения" материала.

Закон Гука для изотропного материала

Напряжения и деформации связаны линейными соотношениями

$$\begin{pmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{pmatrix}=\frac{E}{1-\nu^2}\begin{pmatrix}1 & \nu & 0\\ \nu & 1 & 0\\ 0 & 0 & \frac{1-\nu}{2}\end{pmatrix}\begin{pmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ 2\varepsilon_{xy}\end{pmatrix} \tag{41}$$

где E — модуль Юнга, ν — коэффициент Пуассона.

Геометрическая интерпретация

1. Тензор напряжений

- Действует на площадке с нормалью $\vec{n}=(n_x, n_y)$:

$$\vec{T}=\sigma\cdot\vec{n}=\begin{pmatrix}\sigma_{xx}n_x+\sigma_{xy}n_y\\ \sigma_{xy}n_x+\sigma_{yy}n_y\end{pmatrix}$$

- Инварианты: $\sigma_{xx}+\sigma_{yy}$ (гидростатическое давление), $\sigma_{xx}\sigma_{yy}-\sigma_{xy}^2$ (касательные напряжения).

2. Тензор деформаций

- Изменение длины элемента dl

$$\frac{(dl')^2-(dl)^2}{2}=\varepsilon_{xx}dx^2+\varepsilon_{yy}dy^2+2\varepsilon_{xy}dx\,dy \tag{42}$$

- Изменение угла между осями: $\gamma_{xy}=2\varepsilon_{xy}$.

Уравнения равновесия + Закон Гука + Уравнение совместности образуют замкнутую систему для определения 8 неизвестных ($\sigma_{xx}, \sigma_{yy}, \sigma_{xy}, \varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy}, u_x, u_y$).

Функция Эри $\Phi(x, y)$ автоматически удовлетворяет уравнениям равновесия, а условие $\nabla^4\Phi=0$ обеспечивает совместность деформаций. Для полиномиальных решений функция Эри сводит задачу к поиску коэффициентов, удовлетворяющих бигармоническому уравнению и граничным условиям.

Это основа для математического моделирования в механике материалов, аэрокосмической инженерии и физике твердого тела.

Решение двумерных задач линейной теории упругости с использованием функции Эри

Рассмотрим линейно-упругое тело в условиях плоского напряженного состояния (ПНС) или плоского деформированного состояния (ПДС). Уравнения равновесия (без массовых сил) и условие совместности деформаций (Сен-Венана) имеют вид:

Уравнения равновесия

$$\frac{\partial \sigma_{xx}}{\partial x}+\frac{\partial \sigma_{xy}}{\partial y}=0 \tag{43}$$

Уравнение совместности деформаций

$$\frac{\partial^2 \varepsilon_{xx}}{\partial y^2}+\frac{\partial^2 \varepsilon_{yy}}{\partial x^2}=2\frac{\partial^2 \varepsilon_{xy}}{\partial x \partial y} \tag{44}$$

Используя уравнения равновесия (37), условие совместности деформаций (40) и закон Гука для плоского состояния для диагональных компонентов тензора напряжений можно получить соотношения

$$\nabla^2\left(\sigma_{xx}+\sigma_{yy}\right)=0 \text{ (для ПНС)} \tag{45}$$

где $\nabla^2=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}$ - оператор Лапласа.

Функция Эри (Airy Stress Function)

Ключевая идея: Ввести такую скалярную функцию $\Phi\left(x,y\right)$ (функцию Эри), через вторые производные которой автоматически выражаются компоненты напряжения и которая тождественно удовлетворяет уравнениям равновесия

$$\begin{aligned}\sigma_{xx}&=\frac{\partial^2 \Phi}{\partial y^2}\\ \sigma_{yy}&=\frac{\partial^2 \Phi}{\partial x^2}\\ \sigma_{xy}&=-\frac{\partial^2 \Phi}{\partial x \partial y}\end{aligned} \tag{46}$$

Уравнение для функции Эри

Подставим выражения напряжений через Ф в уравнение совместности, выраженное через напряжения. Для обоих случаев (ПНС и ПДС при обычных предположениях) получим одно и то же бигармоническое уравнение

$$\nabla^4\Phi=0 \text{ или } \left(\frac{\partial^4}{\partial x^4}+2\frac{\partial^4}{\partial x^2\partial y^2}+\frac{\partial^4}{\partial y^4}\right)\Phi=0 \tag{47}$$

где $\nabla^4=\nabla^2\left(\nabla^2\right)$ - бигармонический оператор.

Таким образом, задача сводится к отысканию бигармонической функции $\Phi\left(x,y\right)$), удовлетворяющей граничным условиям задачи на контуре тела.

Метод полиномиальных решений

Суть метода: Искать решение бигармонического уравнения $\nabla^4 \Phi = 0$ в виде полиномов от переменных x и y. Это один из самых простых и наглядных методов для решения задач с прямолинейными границами (балки, клинья, пластины с отверстиями, концентраторы напряжений).

Почему полиномы?

1. Операторы ∇^2 и ∇^4 являются линейными дифференциальными операторами с постоянными коэффициентами.

2. Полиномы - простейшие функции, удобные для дифференцирования.

3. Производные полиномов сами являются полиномами (или нулями).

4. Условие $\nabla^4 \Phi = 0$ накладывает ограничения на коэффициенты полинома, делая задачу разрешимой.

5. Полиномиальные решения часто соответствуют фундаментальным состояниям напряжений (чистое растяжение/сжатие, чистый сдвиг, чистый изгиб).

Порядок полинома: Рассмотрим полином Φ степени n

$$\Phi_n(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n-i} c_{ij} x^i y^j, \tag{48}$$

где c_{ij} — коэффициенты, удовлетворяющие условию $\nabla^4 \Phi_n = 0$, $i + j \le n$.

Анализ по степеням:

$n = 0, 1: \Phi_0 = c_{00}, \Phi_1 = c_{10} x + c_{01} y.$

Для них $\nabla^4 \Phi \equiv 0$, но напряжения:

$$\sigma_{xx} = \frac{\partial^2 \Phi}{\partial y^2} \equiv 0,$$
$$\sigma_{yy} = \frac{\partial^2 \Phi}{\partial x^2} \equiv 0, \quad .$$
$$\sigma_{xy} = -\frac{\partial^2 \Phi}{\partial x \partial y} \equiv 0$$

Это состояние нулевых напряжений (может включать жесткое смещение).

$n = 2: \Phi_2 = c_{20} x^2 + c_{11} xy + c_{02} y^2$

$\nabla^4 \Phi_2 = 0$ (так как вторые производные - константы, четвертые - нули).

Напряжения:

$$\sigma_{xx}=\frac{\partial^2\Phi}{\partial y^2}\equiv 2c_{02},$$
$$\sigma_{yy}=\frac{\partial^2\Phi}{\partial x^2}\equiv 2c_{20},$$
$$\sigma_{xy}=-\frac{\partial^2\Phi}{\partial x\partial y}\equiv -c_{11}$$

Физический смысл: Однородное (постоянное) поле напряжений:

- Равномерное растяжение/сжатие вдоль $x\,(c_{20}\neq 0)$.
- Равномерное растяжение/сжатие вдоль $y\,(c_{02}\neq 0)$.
- Чистый сдвиг $(c_{11}\neq 0)$.

$n=3$: $\Phi_3=c_{30}x^3+c_{21}x^2y+c_{12}xy^2+c_{03}y^3$

$\nabla^4\Phi_3=0$ (третьи производные - константы, четвертые - нули).

Напряжения (линейные функции):

$$\sigma_{xx}=\frac{\partial^2\Phi}{\partial y^2}\equiv 2c_{12}x+6c_{03}y,$$
$$\sigma_{yy}=\frac{\partial^2\Phi}{\partial x^2}\equiv 6c_{30}x+2c_{21}y,$$
$$\sigma_{xy}=-\frac{\partial^2\Phi}{\partial x\partial y}\equiv -2c_{21}x-2c_{12}y$$

Физический смысл: Линейно меняющиеся напряжения. Классический пример - чистый изгиб консольной балки постоянного сечения силой на конце. Коэффициенты подбираются под граничные условия.

$n=4$: $\Phi_4=c_{40}x^4+c_{31}x^3y+c_{22}x^2y^2+c_{13}xy^3+c_{04}y^4$

Условие $\nabla^4\Phi_4=0$ НЕ выполняется автоматически! Вычислим:

$$\nabla^2\Phi_4=\frac{\partial^4\Phi_4}{\partial x^4}+2\frac{\partial^4\Phi_4}{\partial x^2\partial y^2}+\frac{\partial^4\Phi_4}{\partial y^4}=24c_{40}+8c_{22}+24c_{04}$$

Чтобы $\nabla^4\Phi_4=\nabla^2(\nabla^2\Phi_4)=0$, необходимо и достаточно, чтобы $\nabla^2\Phi_4=const=K$ (так как $\nabla^2(const)=0$).

Для Φ_4 при $K=0$ это дает:

$$24c_{40}+8c_{22}+24c_{04}=0$$

Свобода выбора: Мы можем выразить один коэффициент через другие. Обычно связывают c_{40},c_{22},c_{04}. Например, $c_{22}=-3c_{40}-3c_{04}$. Коэффициенты c_{31} и c_{13} остаются свободными.

Физический смысл: Более сложные распределения напряжений. Примеры:

- Решение для балки под действием собственного веса (y^3).
- Решение для растяжения пластины с малым круговым отверстием (используется комбинация полиномов и специальных функций).

$n \geq 5$: Аналогично степени 4. Условие $\nabla^4 \Phi_n = 0$ приводит к линейным соотношениям между коэффициентами полинома.

Применение граничных условий

Найденная функция Эри $\Phi(x, y)$ в виде полинома (или суммы полиномов) должна удовлетворять граничным условиям задачи на контуре тела. Граничные условия ставятся обычно в напряжениях или перемещениях.

Граничные условия в напряжениях: На контуре заданы поверхностные силы (нагрузки) $T_x = \sigma_{xx} n_x + \sigma_{xy} n_y$ и $T_y = \sigma_{xy} n_x + \sigma_{yy} n_y$, где n_x, n_y - компоненты вектора нормали к контуру. Так как напряжения выражаются через вторые производные Φ, эти условия преобразуются в условия на Φ и ее производные на границе.

Граничные условия в перемещениях: Требуют предварительного определения поля перемещений u, v путем интегрирования уравнений связи деформаций с перемещениями и закона Гука. Это сложнее.

Процедура решения:

1. Исходя из геометрии и характера нагрузки, предположить подходящую форму полинома для Φ (например, для симметричной балки на изгиб подойдет полином 3-й степени).
2. Убедиться, что полином удовлетворяет $\nabla^4 \Phi = 0$ (для $n \geq 4$ наложить необходимые связи на коэффициенты).
3. Выразить напряжения через Φ.
4. Записать граничные условия на всех участках контура тела.
5. Подставить выражения напряжений в граничные условия. Это даст систему линейных алгебраических уравнений относительно неизвестных коэффициентов полинома.
6. Решить систему и найти коэффициенты.
7. Найти напряжения и, при необходимости, перемещения.

Преимущества и ограничения метода полиномов с функцией Эри

Преимущества:

- Математическая строгость и точность решения.
- Относительная простота для задач с прямолинейными границами и полиномиальными нагрузками.

- Наглядность связи вида полинома с типом напряженного состояния.

- Хорошо подходит для решения задач об изгибе балок, концентрации напряжений у отверстий (в комбинации с другими функциями), клиньях.

Ограничения:

Трудно применим к задачам со сложными криволинейными границами.

Для сложных нагрузок может потребоваться полином очень высокой степени, что делает решение громоздким.

Не всегда очевиден выбор начальной формы полинома.

Заключение:

Функция Эри является мощным инструментом для решения двумерных задач линейной теории упругости, сводя систему уравнений к отысканию одной бигармонической функции. Метод полиномиальных решений позволяет находить точные аналитические решения для широкого класса задач с прямолинейными границами, начиная от простейших однородных напряженных состояний и кончая сложными задачами изгиба и концентрации напряжений. Понимание связи степени полинома функции Эри с характером распределения напряжений является ключевым для успешного применения этого метода.

Рассмотрим несколько примеров решения задач на использование функции Эри с применением sympy.

Пример 1. С помощью функции Эри построить различные полиномиальные решения плоской задачи теории упругости.

```
from sympy import symbols, diff, Eq, solve, expand

# Символьные переменные
x, y = symbols('x y')

# Функция для проверки бигармонического уравнения
def is_biharmonic(phi, x, y):
    return diff(phi, x, 4) + 2*diff(phi, x, 2, y, 2) + diff(phi, y, 4)

# Полином 2-й степени (c20, c11, c02)
c20, c11, c02 = symbols('c20 c11 c02')
phi_2 = c20*x**2 + c11*x*y + c02*y**2
print("Полином 2-й степени:", phi_2)
print("Бигармоническое уравнение:", expand(is_biharmonic(phi_2, x, y)), "\n")

# Полином 3-й степени (c30, c21, c12, c03)
c30, c21, c12, c03 = symbols('c30 c21 c12 c03')
phi_3 = c30*x**3 + c21*x**2*y + c12*x*y**2 + c03*y**3
print("Полином 3-й степени:", phi_3)
print("Бигармоническое уравнение:", expand(is_biharmonic(phi_3, x, y)), "\n")
```

```
# Полином 4-й степени (c40, c31, c22, c13, c04)
c40, c31, c22, c13, c04 = symbols('c40 c31 c22 c13 c04')
phi_4 = c40*x**4 + c31*x**3*y + c22*x**2*y**2 + c13*x*y**3 + c04*y**4
biharmonic_4 = is_biharmonic(phi_4, x, y)
print("Полином 4-й степени:", phi_4)
print("Бигармоническое уравнение:", expand(biharmonic_4))

# Решаем уравнение бигармоничности: 24*c40 + 8*c22 + 24*c04 = 0
eq_bi4 = Eq(24*c40 + 8*c22 + 24*c04, 0)
c22_sol = solve(eq_bi4, c22)[0]
print("\nОграничение для c22:", c22_sol)
phi_4_const = phi_4.subs(c22, c22_sol)
print("Полином 4-й степени с ограничением:", phi_4_const)

# Вычисляем компоненты напряжений
def stress_components(phi, x, y):
    sig_xx = diff(phi, y, 2)
    sig_yy = diff(phi, x, 2)
    sig_xy = -diff(phi, x, y)
    return sig_xx, sig_yy, sig_xy

sig_xx_4, sig_yy_4, sig_xy_4 = stress_components(phi_4_const, x, y)
print("\nКомпоненты напряжений для полинома 4-й степени:")
print("σ_xx =", sig_xx_4)
print("σ_yy =", sig_yy_4)
print("σ_xy =", sig_xy_4)

# Полином 5-й степени (c50, c41, c32, c23, c14, c05)
c50, c41, c32, c23, c14, c05 = symbols('c50 c41 c32 c23 c14 c05')
phi_5 = c50*x**5 + c41*x**4*y + c32*x**3*y**2 + c23*x**2*y**3 + c14*x*y**4 + c05*y**5
biharmonic_5 = is_biharmonic(phi_5, x, y)
print("\nПолином 5-й степени:", phi_5)
print("Бигармоническое уравнение:", expand(biharmonic_5))

# Решаем уравнения бигармоничности:
# Коэффициенты при x: 120*c50 + 24*c32 + 24*c14 = 0
# Коэффициенты при y: 24*c41 + 24*c23 + 120*c05 = 0
eq_x = Eq(120*c50 + 24*c32 + 24*c14, 0)
eq_y = Eq(24*c41 + 24*c23 + 120*c05, 0)
sol_5 = solve([eq_x, eq_y], [c32, c23])
print("\nОграничения для коэффициентов:")
print("c32 =", sol_5[c32])
print("c23 =", sol_5[c23])
phi_5_const = phi_5.subs({
    c32: sol_5[c32],
    c23: sol_5[c23]
})
print("Полином 5-й степени с ограничениями:", phi_5_const)
```

В программе вычисляются полиномы функции Эри (48) для $n \leq 5$. Результат работы программы:

Полином 2-й степени: c02*y**2 + c11*x*y + c20*x**2

Бигармоническое уравнение: 0

Полином 3-й степени: c03*y**3 + c12*x*y**2 + c21*x**2*y + c30*x**3

Бигармоническое уравнение: 0

Полином 4-й степени: c04*y**4 + c13*x*y**3 + c22*x**2*y**2 + c31*x**3*y + c40*x**4

Бигармоническое уравнение: 24*c04 + 8*c22 + 24*c40

Ограничение для c22: -3*c04 - 3*c40

Полином 4-й степени с ограничением: c04*y**4 + c13*x*y**3 + c31*x**3*y + c40*x**4 + x**2*y**2*(-3*c04 - 3*c40)

Компоненты напряжений для полинома 4-й степени:

σ_xx = 6*(2*c04*y**2 + c13*x*y - x**2*(c04 + c40))

σ_yy = 6*(c31*x*y + 2*c40*x**2 - y**2*(c04 + c40))

σ_xy = -3*(c13*y**2 + c31*x**2 - 4*x*y*(c04 + c40))

Полином 5-й степени: c05*y**5 + c14*x*y**4 + c23*x**2*y**3 + c32*x**3*y**2 + c41*x**4*y + c50*x**5

Бигармоническое уравнение: 120*c05*y + 24*c14*x + 24*c23*y + 24*c32*x + 24*c41*y + 120*c50*x

Ограничения для коэффициентов:

c32 = -c14 - 5*c50

c23 = -5*c05 - c41

Полином 5-й степени с ограничениями: c05*y**5 + c14*x*y**4 + c41*x**4*y + c50*x**5 + x**3*y**2*(-c14 - 5*c50) + x**2*y**3*(-5*c05 — c41)

Пример 2. Найти распределение напряжений и деформаций в консольно закрепленной балке узкого прямоугольного сечения, нагруженной на конце силой P.

Рассмотрим консоль, имеющую узкое прямоугольное поперечное сечение единичной толщины (рис. 7), изгибаемую силой P, приложенной на конце.

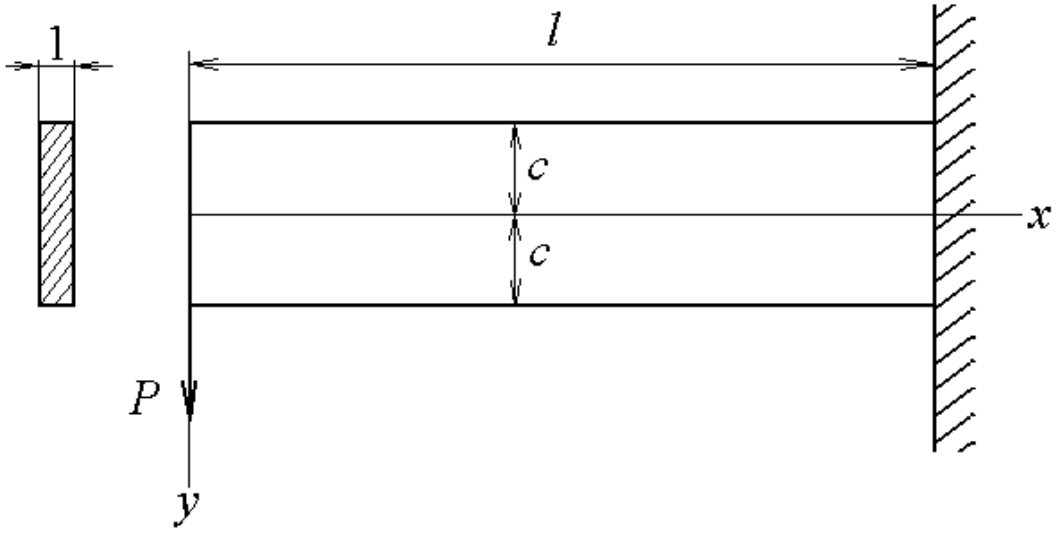


Рисунок 7 - Схема нагружения консольной балки

Верхняя и нижняя грани консоли свободны от нагрузки, на торце $x=0$ распределены касательные усилия, имеющие результирующую силу *P*. Этим условиям нагружения можно удовлетворить, выбрав надлежащую комбинацию напряжений чистого сдвига с напряжениями, вычисляемыми с помощью функции Эри в виде полинома четвертого порядка, рассмотренную в предыдущем примере. В соответствии с этим определим компоненты напряжения следующим образом

$$\begin{gathered}\sigma_{xx}=a_3x^2+a_4xy-\left(2a_3+a_1\right)y^2\\ \sigma_{yy}=a_1x^2+a_2xy+a_3y^2\\ \sigma_{xy}=-a_0-\frac{a_2}{2}x^2-2a_{3xy}-\frac{a_4}{2}y^2\end{gathered}, \tag{49}$$

где a_0,a_1,a_2,a_3,a_4 — коэффициенты полинома функции Эри для $n=4$, с учетом их линейной зависимости.

Для вычисления неизвестных коэффициентов используем граничные условия. Верхняя часть балки при $y=c$ свободна от напряжений. Это позволяет записать следующие уравнения

$$\begin{gathered}a_1x^2+a_2cx+a_3c^2=0\\ a_1x^2-a_2cx+a_3c^2=0\\ -a_0-a_2\frac{x^2}{2}-2a_3cx-a_4\frac{c^2}{2}=0\end{gathered} \tag{50}$$

Еще одно уравнение дает условие отсутствия касательных напряжений на нижней части балки, при $y=-c$

$$-a_0-a_2\frac{x^2}{2}+2a_3cx-a_4\frac{c^2}{2}=0 \tag{51}$$

На нагруженном конце балки сумма касательных усилий, распределенных по торцу, должна быть равна -*P* (знак "минус" возникает по правилу знаков для касательных напряжений). Отсюда получаем еще одно уравнение

$$\int_{-c}^{c}\sigma_{xy}dy=-P \tag{52}$$

что дает

$$-a_4\frac{c^3}{3}+2c\left(-a_0-a_2\frac{x^2}{2}\right)+P=0 \tag{53}$$

Таким образом получаем систему пяти уравнений (50),(51),(53), решая которую получаем значения коэффициентов

$$\begin{aligned} a_0&=3\frac{P}{4c}\\ a_1&=0\\ a_2&=0\\ a_3&=0\\ a_4&=-3\frac{P}{2c^3} \end{aligned} \quad . \tag{54}$$

Подставляя коэффициенты в уравнения для напряжений (49) получаем

$$\begin{aligned} \sigma_{xx}&=-3P\frac{xy}{2c^3}\\ \sigma_{yy}&=0\\ \sigma_{xy}&=3P\frac{y^2-c^2}{4c^3} \end{aligned} \quad . \tag{55}$$

Заметим, что с учетом того, что $\frac{2}{3c^3}$ - это момент инерции поперечного сечения консоли, полученный результат совпадает с элементарным решением, которое дается в курсах сопротивления материалов.

Введем обозначения для компонентов тензора деформации ε_{11}, ε_{22} и ε_{12} соответственно и вычислим их по закону Гука

$$\varepsilon_{11}=\frac{\sigma_{11}}{E}$$
$$\varepsilon_{22}=-\nu\frac{\sigma_{11}}{E} \quad . \tag{56}$$
$$\varepsilon_{12}=\sigma_{12}\frac{(1+\nu)}{E}$$

В результате получим

$$\varepsilon_{11}=-3\frac{P x y}{2 E c^3}$$
$$\varepsilon_{22}=3\frac{P \nu x y}{2 E c^3} \quad . \tag{57}$$
$$\varepsilon_{12}=3\frac{P(-c^2+y^2)(\nu+1)}{4 E c^3}$$

Для вычисления компонентов вектора перемещения необходимо проинтегрировать выражения компонентов тензора деформации исходя из их определения

$$\varepsilon_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right). \tag{58}$$

$$u_1=\int \varepsilon_{11}\,dx=-3\frac{P x^2 y}{4 E c^3}+f1(y)$$
$$u_2=\int \varepsilon_{22}\,dy=3\frac{P \nu x y^2}{4 E c^3}+f2(x) \tag{59}$$

Поскольку определения компонентов тензора деформации представляют собой дифференциальные уравнения в частных производных, то при их интегрировании возникают произвольные функции $f1(y)$ и $f2(x)$. Для их нахождения используем условие для компоненты тензора деформации ε_{12} из определения (58)

$$\frac{\partial u_1}{\partial y}+\frac{\partial u_2}{\partial x}=2\varepsilon_{12} \tag{60}$$

Подставляя в это соотношение выражения компоненты вектора перемещения (59) и ε_{12} из (57) получим

$$-3\frac{P x^2}{4 E c^3}+\frac{df1(y)}{dy}+3\frac{P \nu y^2}{4 E c^3}-3\frac{P y^2(\nu+1)}{2 E c^3}+\frac{df2(x)}{dx}=$$
$$-3\frac{P c^2(\nu+1)}{2 E c^3} \tag{61}$$

Это уравнение можно переписать в виде

$$F(x)+G(y)=K, \tag{62}$$

где:

$$F(x)=-3\frac{P x^2}{4 E c^3}+\frac{df2(x)}{dx}$$

$$G(y)=3\frac{P v y^2}{4 E c^3}-3\frac{P y^2(v+1)}{2 E c^3}+\frac{df1(y)}{dy} \tag{63}$$

$$K=-3\frac{P(v+1)}{2 E c}$$

В уравнении (62) сумма функции $F(x)$, зависящей только от переменной x, и функции $G(y)$, зависящей только от y, равна константе K. Это возможно, только в том случае, если каждая из этих функций также равна некоторой константе, которые обозначим как C_1 и C_2, соответственно. В результате для производных функций $f1, f2$ получаем уравнения

$$\frac{df2(x)}{dx}=3\frac{P x^2}{4 E c^3}+C_1 \tag{64}$$

$$\frac{df1(y)}{dy}=\frac{6 P y^2+3 P v y^2}{4 E c^3}+C_2 \tag{65}$$

Интегрируя эти уравнения по x и по y соответственно получим

$$f2(x)=\frac{P x^3}{4 E c^3}+C_1 x+C_3 \tag{66}$$

$$f1(y)=\frac{2 P y^3+P v y^3}{4 E c^3}+C_2 y+C_4 \tag{67}$$

Подставим найденные функции в выражения для компонентов вектора перемещения (59)

$$u_1=-3\frac{P x^2 y}{4 E c^3}+\frac{2 P y^3+P v y^3}{4 E c^3}+C_2 y+C_4 \tag{68}$$

$$u_2=3\frac{P v x y^2}{4 E c^3}+\frac{P x^3}{4 E c^3}+C_1 x+C_3 \tag{69}$$

Найденные выражения содержат четыре неизвестных постоянных интегрирования C_1, C_2, C_3, C_4. Их необходимо определить из граничных условий. В точке закрепления $x=l, y=0$ компоненты вектора перемещения равны нулю. Тогда из выражения для u_1 следует, что $C_4=0$, а из выражения для u_2 можно выразить константу C_3 через C_1. В результате в выражение для u_2 будет входить уже только константа C_1. Для вычисления константы C_1 учтем, что в точке закрепления консоли $x=l, y=0$ горизонтальная ось балки фиксирована, то есть

$\frac{\partial u_2}{\partial x}=0$. Из этого уравнения можно выразить C_1. Константу C_2 можно выразить из условия $C_1+C_2=-K$.

В результате окончательные выражения для компонентов вектора перемещений принимают вид

$$u_1(x,y)=P\,y\,\frac{3l^2-1^2c^2(v+1)-3x^2+y^2(v+2)}{4Ec^3}$$
$$u_2(x,y)=P\,\frac{2l^3-3l^2x+3v\,x\,y^2+x^3}{4Ec^3} \quad (70)$$

Ось балки в точке нагружения сместится на величину

$$u_2(0,0)=\frac{Pl^3}{2Ec^3}. \quad (71)$$

Это совпадает со значением, которое обычно получается в элементарных курсах сопротивления материалов.

На рисунках 8-10 приведено распределение нормальных и касательных напряжений в сечении балки, а также зависимость прогиба в точке нагружения от толщины балки c для конкретных числовых параметров.

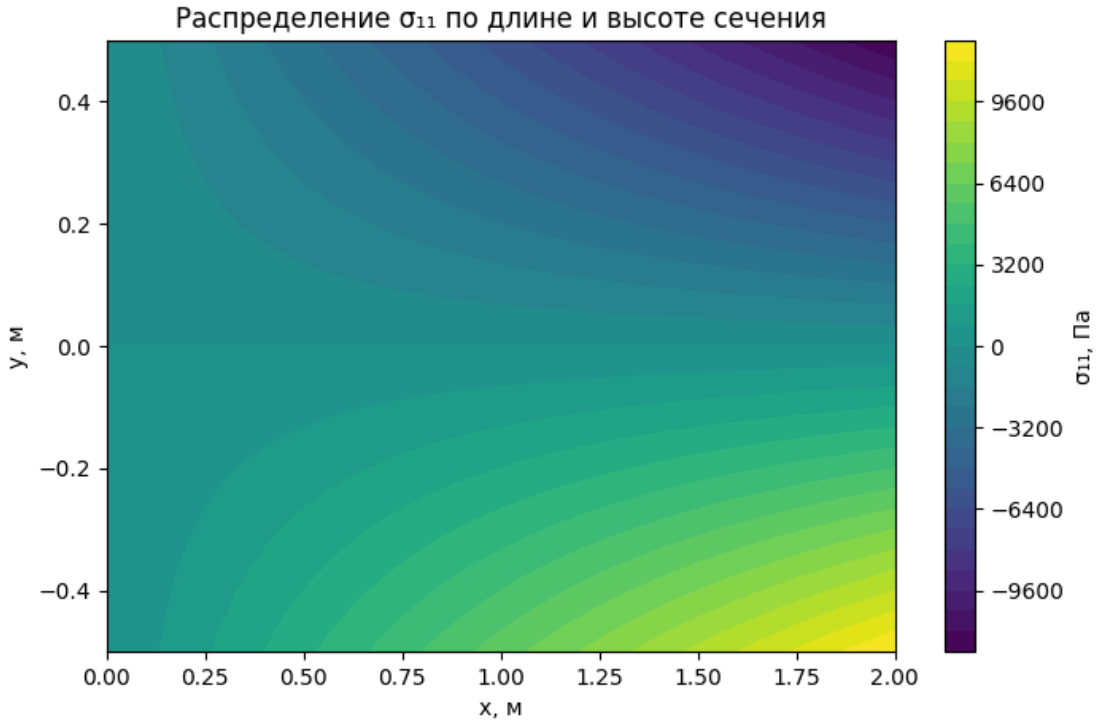


Рисунок 8 - Распределение нормального напряжения σ_{11} по сечению балки

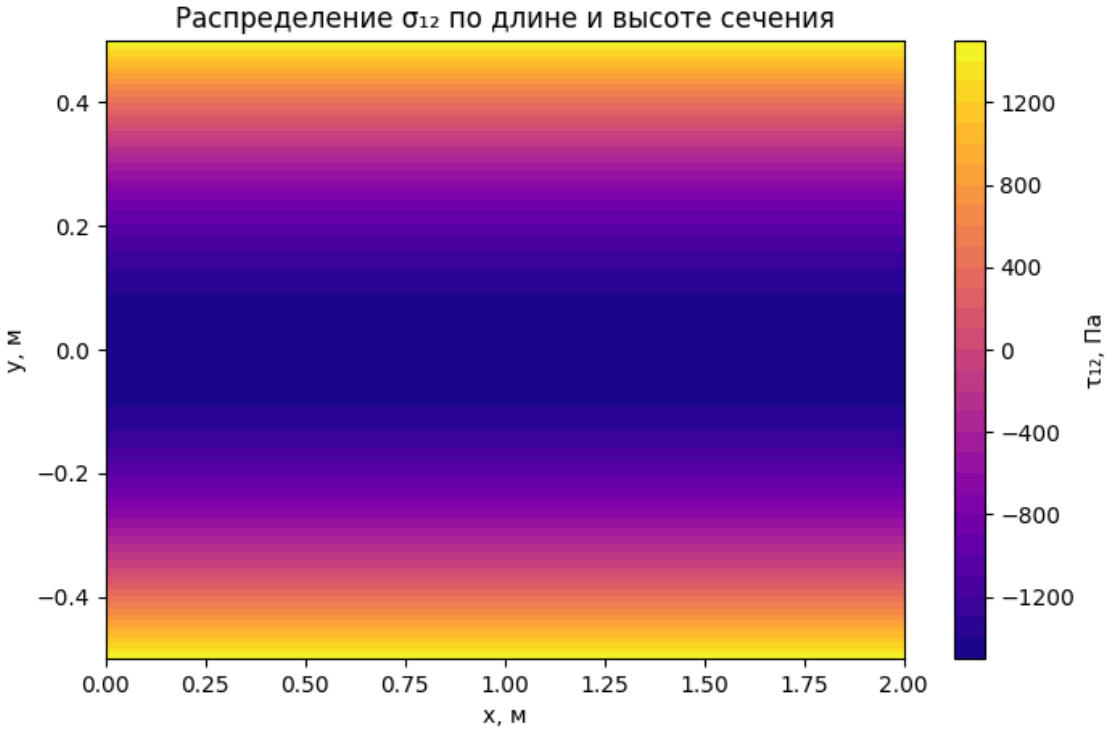


Рисунок 9 - Распределение касательного напряжения σ_{12} по сечению балки

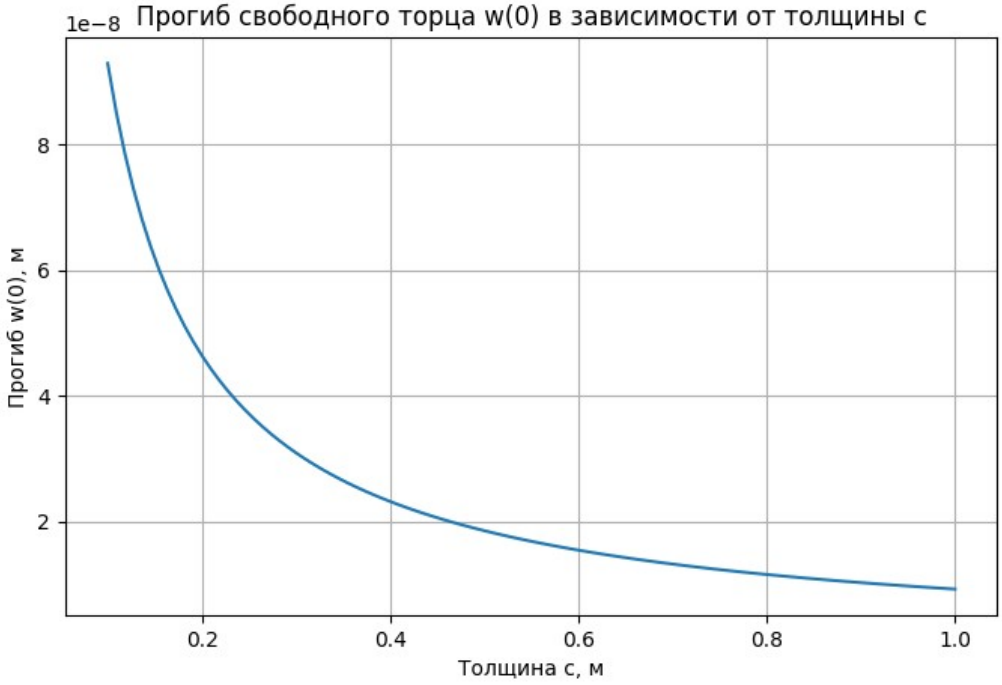


Рисунок 10 - Зависимость прогиба балки в точке нагружения от толщины балки c

Ниже приведен код программы с решением данной задачи

```
import sympy as sp
# ---------------------------------------------------------------
# 1. Символы и параметры
# ---------------------------------------------------------------
x, y, c, l = sp.symbols('x y c l', real=True)           # координаты + размеры
P, E, nu   = sp.symbols('P E nu', positive=True)        # нагрузка и константы
a0, a1, a2, a3, a4 = sp.symbols('a0 a1 a2 a3 a4')       # коэффициенты поля σ
# ---------------------------------------------------------------
# 2. Полином четвёртого порядка для напряжений
# ---------------------------------------------------------------
sig11 = a3*x**2 + a4*x*y - (2*a3 + a1)*y**2             # σ11
sig22 = a1*x**2 + a2*x*y + a3*y**2                      # σ22
sig12 = -a0 - a2*x**2/2 - 2*a3*x*y - a4*y**2/2          # σ12
# ---------------------------------------------------------------
# 3. Граничные условия на гранях y = ±c и торце x = 0
# ---------------------------------------------------------------
eqs = [
    sig22.subs(y,  c),                                  # σ22(c)=0
    sig22.subs(y, -c),                                  # σ22(-c)=0
    sig12.subs(y,  c),                                  # σ12(c)=0
    sig12.subs(y, -c),                                  # σ12(-c)=0
    sp.integrate(sig12, (y, -c, c)) + P                 # ∫σ12dy = −P
]
# Решаем 5 уравнений → a0…a4
solσ = sp.solve(eqs, [a0, a1, a2, a3, a4], dict=True)[0]
print(solσ)
# Подставляем решения
sig11 = sp.simplify(sig11.subs(solσ))
sig22 = 0                                               # стало нулём
sig12 = sp.simplify(sig12.subs(solσ))
print(f"\nσ11 = {sig11}\nσ22 = {sig22}\nσ12 = {sig12}")
# ---------------------------------------------------------------
# 4. Деформации (плоское напряжённое состояние)
# ---------------------------------------------------------------
eps11 =  sig11 / E
eps22 = -nu*sig11 / E
eps12 =  sig12*(1+nu) / E       # здесь ε12 = γ12/2
print(f"\nε11 = {eps11}\nε22 = {eps22}\nε12 = {eps12}")
# ---------------------------------------------------------------
# 5. Интегрируем ε → перемещения u1,u2 с произвольными f1(y),f2(x)
# ---------------------------------------------------------------
f1 = sp.Function('f1')
f2 = sp.Function('f2')
u1 = sp.integrate(eps11, x) + f1(y)
u2 = sp.integrate(eps22, y) + f2(x)
print(sp.simplify(u1))
print(sp.simplify(u2))
```

```
# ---------------------------------------------------------------
# 6. Совмещаем с условием γ12: ∂u1/∂y + ∂u2/∂x = 2ε12
#    Разделяем переменные → появляются константы C1,C2
# ---------------------------------------------------------------
C1, C2 = sp.symbols('C1 C2')
df2_dx =  3*P*x**2/(4*E*c**3) + C1                    # F(x)=C1
df1_dy = -(6*P*c**2*(1+nu) - 3*P*nu*y**2 - 6*P*y**2) \
        /(4*E*c**3) + C2                              # G(y)=C2
# ---------------------------------------------------------------
# 7. Интегрируем функции и добавляем интегр. константы C3,C4
# ---------------------------------------------------------------
C3, C4 = sp.symbols('C3 C4')
f2 = sp.integrate(df2_dx, x) + C3
f1 = sp.integrate(df1_dy, y) + C4
u1 = -3*P*x**2*y/(4*E*c**3) + f1
u2 =  3*P*nu*x*y**2/(4*E*c**3) + f2
# ---------------------------------------------------------------
# 8. Граничные условия в заделке (x = l, y = 0)
# ---------------------------------------------------------------
eq_bc1 = sp.Eq(u1.subs({x: l, y: 0}), 0)              # u1(l,0)=0
eq_bc2 = sp.Eq(u2.subs({x: l, y: 0}), 0)              # u2(l,0)=0
eq_bc3 = sp.Eq(sp.diff(u2, x).subs({x: l, y: 0}), 0)     # ∂u2/∂x(l,0)=0
# Решаем первые три ГУ → C1,C3,C4
solC = sp.solve([eq_bc1, eq_bc2, eq_bc3], [C1, C3, C4])
# Константа K из уравнения F+G = –K
K   = 3*P*(1+nu)/(2*E*c)
C1_ = solC[C1]
C2_ = -K - C1_                                        # связь C1 + C2 = -K
C3_ = solC[C3]
C4_ = solC[C4]
# ---------------------------------------------------------------
# 9. Окончательные выражения
# ---------------------------------------------------------------
u1 = sp.simplify(u1.subs({C1: C1_, C2: C2_, C3: C3_, C4: C4_}))
u2 = sp.simplify(u2.subs({C1: C1_, C2: C2_, C3: C3_, C4: C4_}))
# Сдвиг оси в точке нагружения (x = 0, y = 0)
w0 = sp.simplify(u2.subs({x: 0, y: 0}))
# ---------------------------------------------------------------
# 10. Вывод
# ---------------------------------------------------------------
print("\nПеремещения:")
print("u1(x,y) =", u1)
print("u2(x,y) =", u2)
print("\nПрогиб свободного торца w(0) =", w0)   # →  P*l**3/(2*E*c**3)
import numpy as np
import matplotlib.pyplot as plt
# Числовые параметры
params = {'P': 1000, 'c': 0.5, 'l': 2.0, 'E': 2.1e11, 'nu': 0.3}
```

```
# Сетка по x и y для балки
xv = np.linspace(0, params['l'], 100)
yv = np.linspace(-params['c'], params['c'], 100)
X, Y = np.meshgrid(xv, yv)
# sigma_11(x, y)
sig11 = -3 * params['P'] * X * Y / (2 * params['c']**3)
plt.figure(figsize=(8,5))
plt.contourf(X, Y, sig11, levels=30, cmap='viridis')
plt.colorbar(label='σ₁₁, Па')
plt.title('Распределение σ₁₁ по длине и высоте сечения')
plt.xlabel('x, м')
plt.ylabel('y, м')
plt.show()
# sigma_12(y)
sig12 = -3 * params['P'] / (4 * params['c']) * (1 - 2*Y**2 / params['c']**2)
plt.figure(figsize=(8,5))
plt.contourf(X, Y, sig12, levels=30, cmap='plasma')
plt.colorbar(label='τ₁₂, Па')
plt.title('Распределение σ₁₂ по длине и высоте сечения')
plt.xlabel('x, м')
plt.ylabel('y, м')
plt.show()
# w(0) как функция c
c_vals = np.linspace(0.1, 1.0, 100)
w0_c = 3 * params['P'] * params['l'] * (1 + params['nu']) / (4 * params['E'] * c_vals)
plt.figure(figsize=(8,5))
plt.plot(c_vals, w0_c)
plt.title('Прогиб свободного торца w(0) в зависимости от толщины c')
plt.xlabel('Толщина c, м')
plt.ylabel('Прогиб w(0), м')
plt.grid(True)
plt.show()
```

1.2. Динамические детерминированные модели

Динамические детерминированные модели описывают эволюцию системы во времени, где будущее состояние однозначно определяется текущим состоянием и управляющими воздействиями (отсутствие случайности). Дополнительная независимая переменная — время, возникающая в динамических детерминированных моделях, в отличие от статических, в зависимости от постановки задачи может принимать непрерывные значения (модели с непрерывным временем, $t \in R$) или дискретные значения (модели с дискретным временем, $t = 0, 1, 2, \ldots$). Сфера применения таких моделей весьма широка: физика (механика, термодинамика), химия (кинетика), биология (популяционная динамика), экология, экономика (макроэкономические модели), инженерные системы (управление, электрические цепи) и др. Основные задачи, решаемые с применением динамических детерминированных моделей, это:

прогнозирование, анализ устойчивости, поиск равновесных состояний, оптимизация траекторий.

1.2.1. Основные понятия и классификация

1.2.1.1. Состояние системы

Состояние системы – это минимальный набор переменных (величин), однозначно характеризующий систему в данный конкретный момент времени и содержащий всю необходимую информацию для предсказания ее поведения в будущем, при известных законах эволюции (уравнениях модели) и будущих входных воздействиях.

Пояснения:

- "Минимальный набор": Состояние включает только те переменные, без знания которых невозможно предсказать будущее. Добавление лишних переменных усложняет модель без пользы.
- "Характеризует в данный момент": Состояние – это "мгновенный снимок" системы. Оно фиксирует текущие свойства системы, такие как положение, количество, уровень, концентрация и т.д.
- "Вся необходимая информация для предсказания будущего": Это самое важное свойство. Знание текущего состояния и будущих входов (управлений, внешних сил) полностью определяет эволюцию системы согласно ее динамическим законам (дифференциальным или разностным уравнениям). Состояние "впитывает" в себя всю предысторию системы, релевантную для ее будущего развития. Прошлое влияет на будущее только через текущее состояние.

Вектор состояния: Состояние удобно представлять как вектор $X(t)=[x_1(t),x_2(t),...,xn(t)]^T$, где $x_i(t)$ – это отдельные переменные состояния. Размерность вектора n называется порядком системы. Изменение состояния во времени (dX/dt для непрерывного времени или $X(t+1)$) для дискретного) описывается уравнениями состояния (динамическими уравнениями) модели.

Примеры переменных состояния в различных областях:

1. Механика материальной точки:

Переменные состояния: Координата положения $(x(t),y(t),z(t))$и компоненты вектора скорости $(v_x(t),v_y(t),v_z(t))$. Зная текущее положение и скорость точки (состояние) и будущие силы (входы), второй закон Ньютона ($F=m\,a$, где $a=dv/dt$) позволяет однозначно определить ее ускорение, а значит, и ее положение и скорость в любой будущий момент времени. Только координаты (x,y,z) для этого недостаточно – две точки могут иметь одинаковое положение, но разную скорость (и разное будущее движение). Только скорости тоже недостаточно.

2. Динамика популяции (экология, биология):

Переменная состояния: Численность (или плотность) популяции в данный момент времени ($N(t)$). Знание текущей численности $N(t)$ (и параметров модели, таких как рождаемость и смертность) позволяет предсказать численность в ближайшем будущем $N(t+\Delta t)$ (например, по уравнению $dN/dt = r\,N$ для экспоненциального роста или $dN/dt = r\,N(1-N/K)$ для логистического роста). Прошлые значения численности влияют на будущее только через текущее значение $N(t)$.

3. Химическая кинетика (реактор):

Переменные состояния: Концентрации участвующих в реакции веществ в данный момент времени ($[A](t), [B](t), [C](t), \ldots$). Скорость химической реакции зависит от текущих концентраций реагентов (закон действующих масс). Зная текущие концентрации (состояние) и константы скоростей реакций, можно однозначно определить, как эти концентрации будут изменяться со временем (решить систему ОДУ типа $d[A]/dt = -k_1[A][B] + k_2[C]$).

4. Простая экономическая модель (капитал):

Переменная состояния: Величина капитала (или фондов) в данный момент времени ($K(t)$). Динамика капитала часто описывается уравнениями, где скорость изменения капитала (dK/dt или ΔK) зависит от текущего уровня капитала (например, через инвестиции, пропорциональные K, или амортизацию) и внешних вложений (входов). Знание $K(t)$ позволяет прогнозировать будущий размер капитала при известной инвестиционной политике.

Выбор переменных состояния не уникален. Для одной и той же системы можно выбрать разные (но эквивалентные) наборы переменных состояния. Например, в механике вместо (x, v_x) можно было бы использовать (x, p_x), где $p_x = m\,v_x$ – импульс. Однако понятие минимального набора и способности предсказывать будущее остается ключевым критерием.

Вывод: Понятие состояния системы является фундаментальным для построения и анализа динамических моделей. Оно формализует "память" системы и служит отправной точкой для описания ее эволюции во времени с помощью дифференциальных или разностных уравнений.

1.2.1.2. Пространство состояний (Фазовое пространство)

Пространство состояний (также часто называемое *фазовым пространством*) – это абстрактное математическое пространство, каждая точка которого однозначно соответствует одному возможному состоянию динамической системы. Осями этого пространства являются переменными состояния системы.

Пояснения:

1. Геометрическое представление состояния: Если состояние системы описывается вектором $X(t) = [x_1(t), x_2(t), \ldots, xn(t)]^T$, то пространство состояний – это n-мерное пространство, где:

- Каждая ось (координата) соответствует одной переменной состояния $(x_1, x_2, \ldots, x_n)$.

- Конкретное состояние системы в момент времени t представляется точкой в этом пространстве с координатами $(x_1(t), x_2(t), \ldots, x_n(t))$.

2. Эволюция = Движение точки: По мере изменения состояния системы во времени (в соответствии с динамическими уравнениями), изображающая точка *движется* в пространстве состояний. Путь, который описывает эта точка, называется фазовой траекторией (или просто траекторией) системы.

3. "Фазовое" vs "Пространство состояний": Термин "фазовое пространство" исторически сложился в физике (особенно в классической механике и термодинамике) и часто подразумевает определенный выбор переменных состояния (например, координаты и импульсы). "Пространство состояний" – более общий термин, используемый в теории систем, математическом моделировании и других дисциплинах, подчеркивающий его фундаментальную роль как области, где задано состояние. В контексте математического моделирования эти термины часто используются как синонимы.

4. Размерность: Размерность пространства состояний равна порядку системы n – количеству независимых переменных состояния, необходимых для однозначного описания системы.

5. Ценность концепции:

- Визуализация: Позволяет геометрически представить все возможные состояния системы и их эволюцию во времени.
- Анализ: Становится основой для мощных методов анализа динамики системы без явного решения уравнений (например, построение фазовых портретов, анализ устойчивости, поиск аттракторов).
- Общность: Предоставляет единый язык для описания и сравнения самых разных динамических систем, независимо от их физической природы.

Примеры пространств состояний:

1. Механика материальной точки (движение вдоль прямой):

- Переменные состояния: Положение $x(t)$, Скорость $v(t)$.

- Пространство состояний: Двумерная плоскость (x, v). Это и есть фазовая плоскость.

- Траектория: Кривая на этой плоскости, показывающая, как изменяются координата x и скорость v точки с течением времени. Например, для гармонического осциллятора траектория – эллипс.

В классической механике фазовое пространство часто определяют через координаты q и импульсы $p = mv$. Это эквивалентное представление.

2. Динамика популяции (логистический рост):

- Переменная состояния: Численность популяции $N(t)$.

- Пространство состояний: Одномерная прямая (ось N).

- Траектория: Движение точки вдоль этой прямой по мере роста или убывания популяции. Начальная точка – $N(0)$, конечная (устойчивое состояние) – $N=K$ (емкость среды).

3. Химическая кинетика (простая реакция A → B):

- Переменные состояния: Концентрация вещества A - $[A](t)$, Концентрация вещества B - $[B](t)$.

- Пространство состояний: Двумерная плоскость $([A],[B])$.

- Траектория: Кривая на этой плоскости, описывающая изменение концентраций A и B со временем. Так как $[A]+[B]=const$ (если объем постоянен), траектория будет прямой линией с наклоном -1. Точка стартует на оси $[A]$ (при $[B]=0$) и движется к оси $[B]$ (при $[A]=0$).

4. Экономическая модель (капитал с инвестициями):

- Переменная состояния: Капитал $K(t)$.

- Пространство состояний: Одномерная прямая (ось K).

- Траектория: Движение точки вдоль оси K от начального уровня $K(0)$ к стационарному уровню K^* (если он существует).

Фазовый портрет: Графическое представление множества типичных траекторий в пространстве состояний, дающее общую картину возможных режимов поведения системы. Показывает направления движения изображающей точки, положения равновесия, их устойчивость, предельные циклы и т.д. Особенно нагляден для двумерных систем (фазовых плоскостей).

Дискретные системы: Для систем с дискретным временем ($t=0,1,2,\ldots$) пространство состояний определяется аналогично. Траектория представляет собой последовательность точек $X(0),X(1),X(2),\ldots$ в этом пространстве, соединенных "переходами". График такой последовательности во времени ($x_t vs t$) – это *временная* диаграмма, а ее представление в пространстве состояний – это *фазовая траектория* (часто в виде "лестницы" или последовательности точек).

Вывод: Пространство состояний (фазовое пространство) является мощнейшим концептуальным и аналитическим инструментом в математическом моделировании. Оно абстрагируется от конкретной природы системы и времени, позволяя сосредоточиться на геометрии динамики: как состояние системы (точка) движется в пространстве всех возможных состояний под действием законов эволюции. Это фундамент для последующего анализа устойчивости, бифуркаций и качественного поведения динамических систем.

1.2.1.3. Классификация моделей по типу времени и состояний

Динамические детерминированные модели систематизируют по двум ключевым признакам:

- Характер временной оси (непрерывное/дискретное время);

- Природа пространства состояний (непрерывное/дискретное множество состояний).

Эта классификация определяет математический аппарат и методы анализа. Основные типы:

1. Непрерывные по времени и состоянию модели

Описание:

- Время: ($t \in \mathbb{R}$) (непрерывный параметр).

- Состояние: вектор ($X(t) \in \mathbb{R}^n$) (непрерывные переменные).

Математический аппарат:

- Обыкновенные дифференциальные уравнения (ОДУ):

$$\frac{dX}{dt} = F(X(t), t)$$

- Уравнения в частных производных (УрЧП) для распределенных систем.

Примеры:

- Движение механической системы (координаты и скорости).

- Диффузия тепла в стержне (уравнение теплопроводности).

- Химическая кинетика (изменение концентраций).

Особенности:

- Анализ через методы решения ОДУ/УрЧП.

- Визуализация в виде гладких траекторий в фазовом пространстве.

2. Дискретные по времени и состоянию модели

Описание:

- Время: ($t = 0, 1, 2, \ldots$) (дискретные шаги).

- Состояние: ($X_t \in \mathbb{Z}^n$) или конечное множество (целочисленные/категориальные значения).

Математический аппарат:

- Конечные автоматы:

$$X_{t+1}=G(X_t,U_t)$$

- Целочисленные разностные уравнения.

Примеры:

- Модель популяции с неперекрывающимися поколениями $(N_{t+1}=r\cdot N_t)$.

- Работа цифрового контроллера (дискретные состояния: "включено"/"выключено").

- Распространение информации в сетях (состояние узла: "информирован"/"не информирован").

Особенности:

- Эволюция описывается переходами между состояниями.

- Траектория — последовательность точек в дискретном пространстве.

3. Дискретные по времени, непрерывные по состоянию модели

Описание:

- Время: дискретное $(t=k\Delta t)$.

- Состояние: непрерывное $(X_t\in\mathbb{R}^n)$.

Математический аппарат:

- Разностные уравнения:

$$X_{t+1}=F(X_t,t)$$

Примеры:

- Экономические модели (капитал в квартальных отчетах:$(K_{t+1}=(1+r)K_t+I_t)$).

- Численное решение ОДУ методом Эйлера:

$$x_{n+1}=x_n+f(x_n)\cdot\Delta t$$

Особенности:

- Возникают при дискретизации непрерывных систем или наблюдении в фиксированные моменты времени.

- Фазовое пространство — $\mathbb{R}^n$, траектория — последовательность точек.

4. Непрерывные по времени, дискретные по состоянию модели

Описание:

- Время: непрерывное ($t \in \mathbb{R}$).

- Состояние: дискретное (конечное или счетное множество).

Математический аппарат:

- Уравнения интенсивностей переходов.

- Детерминированные аналоги марковских цепей.

Примеры:

- Системы массового обслуживания (число заявок в очереди скачкообразно меняется в моменты прихода/ухода).

- Переключение режимов работы двигателя ("холостой ход", "нагрузка").

Особенности:

- Редкий класс для строго детерминированных систем.

- Траектория — кусочно-постоянная функция времени.

5. Особые случаи

Дискретно-событийные модели:

- Состояние меняется в *моменты событий* (не фиксированные интервалы).

- Пример: логистическая цепочка (состояние = уровень запасов; события = поставка/продажа).

Гибридные системы:

- Комбинация непрерывной динамики и дискретных событий.

- Пример: термостат (непрерывное изменение температуры + дискретное включение/выключение нагревателя).

Критерии выбора типа модели

Выбор типа модели определяется (табл. 2):

1. Физической природой системы (непрерывные процессы vs дискретные события).

2. Доступными данными (частота измерений, тип переменных).

3. Целью исследования (прогноз непрерывной динамики vs анализ дискретных состояний).

Наиболее распространены в приложениях:

- Непрерывные модели (ОДУ/УрЧП) для инженерных и естественнонаучных задач.

- Дискретные по времени модели (разностные уравнения) для экономики и биологии.

- Дискретно-событийные подходы для управления сложными технологическими процессами.

Таблица 2. Критерии выбора модели

Фактор	**Непрерывные модели**	**Дискретные модели**
Природа системы	Физические процессы (механика, химия)	Информационные системы, цифровая логика
Данные измерений	Частые аналоговые замеры	Данные в фиксированные моменты
Цель моделирования	Анализ устойчивости, плавная оптимизация	Управление событиями, анализ логики переходов

1.2.2. Модели с непрерывным временем (Дифференциальные уравнения)

1.2.2.1. Обыкновенные дифференциальные уравнения (ОДУ)

ОДУ — фундаментальный инструмент для моделирования динамических систем с непрерывным временем, где эволюция состояния описывается производными по одной независимой переменной (обычно времени). Уравнение связывает неизвестную функцию, её производные и независимую переменную.

2.1.1. Основные понятия

Определение ОДУ:

Уравнение вида

$$F\left(t, y, y', y'', \ldots, y^{(n)}\right)=0, \tag{72}$$

где $y(t)$ — неизвестная функция, $y^{(k)}$ — её k-я производная.

Ключевые характеристики:

1. Порядок уравнения:

Наивысший порядок производной в уравнении.

Пример: $y''+y=0$ — уравнение 2-го порядка.

2. Общее решение:

Семейство функций $y=\phi\left(t, C_1, C_2, \ldots, C_n\right)$, содержащее n произвольных постоянных (где n — порядок ОДУ), удовлетворяющее уравнению при любых значениях постоянных.

Пример: Для $y'=kx$ общее решение: $y=\frac{k}{2}x^2+C$.

3. Частное решение:

Решение, полученное из общего путём подстановки конкретных значений постоянных $C_1,\dots,C_n$.

4. Задача Коши (начальная задача):

Поиск частного решения ОДУ, удовлетворяющего начальным условиям (НУ):

$$y(t_0)=y_0,\ y'(t_0)=y_1,\dots,y^{(n-1)}(t_0)=y_{n-1}.$$

Пример: Для уравнения движения $m\ddot{x}=-kx$ с НУ $x(0)=1$, $\dot{x}(0)=0$.

1.2.2.2. Системы ОДУ первого порядка

Нормальная форма системы:

Система из n уравнений 1-го порядка:

$$\begin{cases} \frac{dx_1}{dt}=f_1(t,x_1,x_2,\dots,x_n) \\ \frac{dx_2}{dt}=f_2(t,x_1,x_2,\dots,x_n) \\ \dots \\ \frac{dx_n}{dt}=f_n(t,x_1,x_2,\dots,x_n) \end{cases} \tag{73}$$

В векторной записи:

$$\frac{d\boldsymbol{X}}{dt}=\boldsymbol{F}(t,\boldsymbol{X}), \tag{74}$$

где $\boldsymbol{X}=(x_1,\dots,x_n)^T$.

Сведение ОДУ высшего порядка к системе 1-го порядка:

Уравнение n-го порядка $y^{(n)}=G(t,y,y',\dots,y^{(n-1)})$ заменой переменных:

$$\begin{gathered} z_1=y \\ z_2=y' \\ \dots \\ z_n=y^{(n-1)} \end{gathered} \tag{75}$$

преобразуется в систему:

$$\begin{aligned} \frac{dz_1}{dt} &= z_2 \\ \frac{dz_2}{dt} &= z_3 \\ &\dots \\ \frac{dz_n}{dt} &= G(t, z_1, z_2, \dots, z_n) \end{aligned} \qquad (76)$$

Пример: Уравнение маятника $\ddot{\theta} + \sin\theta = 0 \rightarrow \begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\sin\theta \end{cases}$.

1.2.2.3. Аналитические методы решения

Применяются для уравнений специальных типов.

1. Разделение переменных:

Для уравнений вида $\frac{dy}{dx} = g(x)h(y).$

Алгоритм:

$$\int \frac{dy}{h(y)} = \int g(x)\,dx + C.$$

Пример: $y' = x\,y \rightarrow \int \frac{dy}{y} = \int x\,dx \rightarrow \ln|y| = \frac{x^2}{2} + C$.

2. Линейные ОДУ с постоянными коэффициентами:

- Однородное уравнение: $y^{(n)} + a_{n-1}y^{(n-1)} + \dots + a_0 y = 0.$

Решение:

Характеристическое уравнение $\lambda^n + a_{n-1}\lambda^{n-1} + \dots + a_0 = 0$.

Корни λ_i определяют базис решений: $e^{\lambda_i t}$, $t^k e^{\lambda t}$ (для кратных корней).

- Неоднородное уравнение: $y^{(n)} + \dots + a_0 y = f(t).$

Решение:

Общее решение = решение однородного уравнения + частное решение неоднородного.

Методы подбора: неопределённых коэффициентов (для $f(t)=e^{\alpha t}$, $\sin\beta t$, многочлены) или вариации постоянных.

Пример. Решим с использованием Sympy уравнение

$$y'+\frac{y}{x}=3x$$

```
from sympy import *
from sympy.abc import x
y = Function('y')(x)
y_ = Derivative(y, x)

DE=y_+y/x-3*x #Определяем переменную DE - левую часть дифференциального уравнения, перенеся в нее все члены заданного уравнения
result = dsolve(DE, y) #Находим общее решение уравнения. Функция dsolve возвращает его в виде объекта Eq - уравнение
print(f'Решение: \ny(x)={result.rhs}\n') # На печать выводим правую часть объекта Eq
print(f'Проверка: {checkodesol(DE, result)}') #Для проверки правильности решения используется функция checkodesol, в которую нужно передать уравнение и решение
```

Результат:

```
Решение:
y(x)=(C1 + x**3)/x

Проверка: (True, 0)
```

Решим данное уравнение с начальным условием $y(1)=2$

```
y = Function('y')
result_ics = dsolve(DE, y(x),ics={y(1):3})
print(f'Решение с начальными условиями: \ny(x)={result_ics.rhs}')
print(f'Проверка: {checkodesol(DE, result_ics)}')
```

Результат:

```
Решение с начальными условиями:
y(x)=(x**3 + 2)/x
Проверка: (True, 0)
```

3. Уравнения в полных дифференциалах:

Для $P(x,y)dx+Q(x,y)dy=0$, где $\frac{\partial P}{\partial y}=\frac{\partial Q}{\partial x}$.

Решение:

Найти функцию $U(x,y)$ такую, что $dU=P\,dx+Q\,dy=0$. Тогда $U(x,y)=C$ — общий интеграл.

1.2.2.4. Численные методы решения

Используются, когда аналитическое решение недостижимо.

1. Метод Эйлера:

- Явная схема:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n),$$

где h — шаг сетки.

Погрешность: $O(h)$. Неустойчив при больших h.

- Неявная схема:

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1}).$$

Требует решения уравнения на каждом шаге (устойчивее явного).

2. Методы Рунге-Кутты:

Повышают точность за счёт вычисления производных в промежуточных точках.

- РК 2-го порядка (модифицированный Эйлер):

$$\begin{cases} k_1 = h \cdot f(t_n, y_n) \\ k_2 = h \cdot f(t_n + h, y_n + k_1) \\ y_{n+1} = y_n + \dfrac{k_1 + k_2}{2} \end{cases}$$

Погрешность: $O(h^2)$.

- РК 4-го порядка (классический):

$$\begin{cases} k_1 = h \cdot f(t_n, y_n) \\ k_2 = h \cdot f(t_n + h/2, y_n + k_1/2) \\ k_3 = h \cdot f(t_n + h/2, y_n + k_2/2) \\ k_4 = h \cdot f(t_n + h, y_n + k_3) \\ y_{n+1} = y_n + \dfrac{k_1 + 2k_2 + 2k_3 + k_4}{6} \end{cases}$$

Погрешность: $O(h^4)$.

3. Устойчивость численной схемы:

Схема устойчива, если малые возмущения начальных данных не нарастают с ростом t .

- Критерий: Для тестового уравнения $y' = \lambda y$ $\mathrm{Re}(\lambda) < 0$ решение должно затухать.

- Область устойчивости: Множество значений $h\lambda$, при которых схема устойчива.

Пример: Явный Эйлер устойчив при $|1 + h\lambda| < 1$.

Итог:

- Аналитические методы эффективны для ограниченных классов ОДУ.

- Численные методы универсальны, но требуют анализа точности и устойчивости.

- Сведение высших порядков к системам 1-го порядка стандартизирует вычислительные подходы.

1.2.2.5. Примеры детерминированных динамических моделей на основе ОДУ

Колебания гармонического осциллятора с затуханием и внешней силой

Рассмотрим систему, описываемую дифференциальным уравнением:

$$\ddot{x} + c\,\dot{x} + kx = F_0 \cos(\omega t), \tag{77}$$

где:

- m = 1.0 кг - масса

- c = 0.1 Н·с/м - коэффициент демпфирования

- k= 1.0 Н/м - коэффициент жесткости пружины

- F_0 = 1.0 Н - амплитуда внешней силы

- ω = 0.8 рад/с - частота внешней силы

- Начальные условия: $x(0) = 0, \dot{x}(0) = 0$

Код решения с использованием SciPy

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# Параметры системы
```

```
m = 1.0     # масса [кг]
c = 0.1     # коэффициент демпфирования [Н·с/м]
k = 1.0     # коэффициент жесткости [Н/м]
F0 = 1.0    # амплитуда внешней силы [Н]
omega = 0.8 # частота внешней силы [рад/с]

# Функция системы ОДУ (преобразование уравнения 2-го порядка в систему 1-го порядка)
def damped_oscillator(t, z):
    x, v = z  # z[0] = x (положение), z[1] = v (скорость)
    dxdt = v
    dvdt = (F0 * np.cos(omega * t) - c * v - k * x) / m
    return [dxdt, dvdt]

# Начальные условия и временной интервал
z0 = [0.0, 0.0]  # начальные условия [x0, v0]
t_span = (0, 100) # временной интервал [0, 100] с
t_eval = np.linspace(0, 100, 2000)  # точки для вычисления решения

# Численное решение
solution = solve_ivp(
    fun=damped_oscillator,
    t_span=t_span,
    y0=z0,
    t_eval=t_eval,
    method='RK45'
)

# Извлечение результатов
t = solution.t
x = solution.y[0]  # положение
v = solution.y[1]  # скорость

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

# График положения
plt.subplot(3, 1, 1)
plt.plot(t, x, 'b', linewidth=1.5)
plt.title('Колебания гармонического осциллятора с затуханием и внешней силой')
plt.ylabel('Положение, x(t) [м]')
plt.grid(True)

# График скорости
plt.subplot(3, 1, 2)
plt.plot(t, v, 'r', linewidth=1.5)
plt.ylabel('Скорость, v(t) [м/с]')
plt.grid(True)
```

```
# Фазовый портрет
plt.subplot(3, 1, 3)
plt.plot(x, v, 'g', linewidth=1.0)
plt.xlabel('Положение, x [м]')
plt.ylabel('Скорость, v [м/с]')
plt.title('Фазовый портрет')
plt.grid(True)

plt.tight_layout()
plt.show()

# Анализ установившегося режима (после 50 секунд)
steady_state_mask = t > 50
steady_state_x = x[steady_state_mask]
steady_state_t = t[steady_state_mask]

# Нахождение амплитуды установившихся колебаний
amplitude = (np.max(steady_state_x) - np.min(steady_state_x)) / 2
print(f"Амплитуда установившихся колебаний: {amplitude:.4f} м")
```

Графики, иллюстрирующие результаты вычислений приведены на рисунке .

Вывод результата расчета амплитуды установившихся колебаний:

Амплитуда установившихся колебаний: 2.8116 м

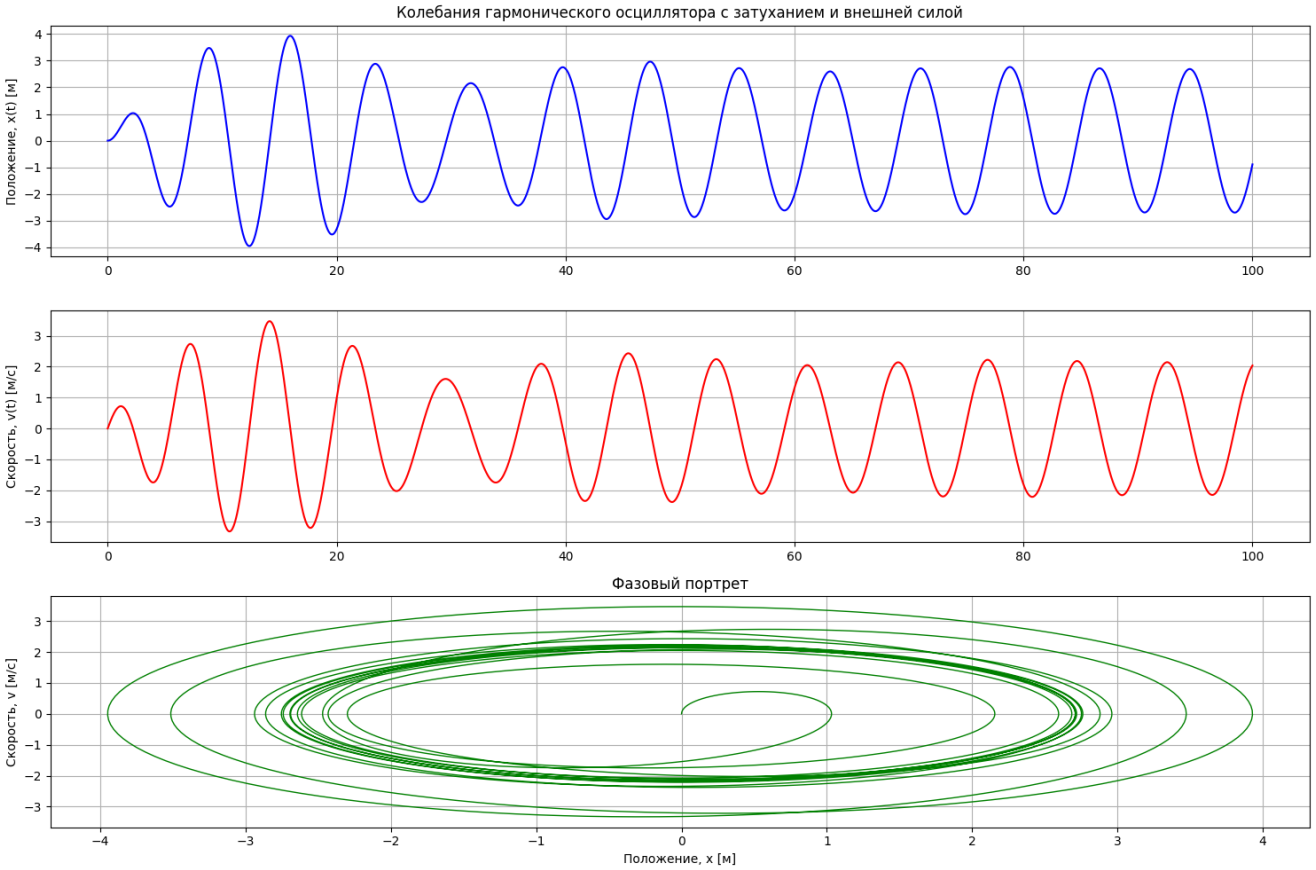


Рисунок 11 - Результаты моделирования колебаний гармонического осциллятора

Физическая интерпретация результатов:

1. Переходный процесс (первые 20-30 секунд):

- Видны сложные колебания с изменяющейся амплитудой

- Происходит "борьба" между собственными колебаниями системы и вынуждающей силой

- Энергия системы нестабильна: происходит диссипация энергии через демпфирование

2. Установившийся режим (после 50 секунд):

- Система переходит в режим вынужденных колебаний

- Колебания происходят с частотой внешней силы $\omega = 0.8$ рад/с

- Амплитуда стабилизируется (примерно 2.8116 м в данном случае)

- Фазовый портрет принимает вид устойчивого предельного цикла

3. Фазовый портрет:

- Спиральное движение к предельному циклу - характерно для демпфированных систем

- Устойчивый замкнутый контур (предельный цикл) соответствует установившимся колебаниям

- Площадь цикла пропорциональна энергии, диссипируемой за период

Теоретическое обоснование:

Для установившегося режима амплитуда колебаний может быть вычислена аналитически:

$$A = \frac{F_0}{\sqrt{(k - \omega^2)^2 + (c\,\omega)^2}}$$

Подставив параметры:

$$A = \frac{1.0}{\sqrt{(1.0 - 1.0 \cdot 0.8^2)^2 + (0.1 \cdot 0.8)^2}} = \frac{1}{\sqrt{(1 - 0.64)^2 + 0.08^2}} = \frac{1}{\sqrt{0.36^2 + 0.08^2}} \approx 2.7116\,м,$$

что близко к результатами численного моделирования.

Особенности модели:

1. Резонанс: При $\omega = \sqrt{k/m} = 1.0$ рад/с амплитуда колебаний была бы максимальной

2. Демпфирование: Уменьшает амплитуду резонанса и ускоряет переход к установившемуся режиму

3. Энергетический баланс: В установившемся режиме работа внешней силы равна энергии, рассеиваемой через демпфирование

Такой осциллятор моделирует широкий класс физических систем: от подвески автомобиля до электрических RLC-цепей.

Модель хищник-жертва Лотки-Вольтерры

Данная модель описывается следующей системой дифференциальных уравнений:

$$\begin{aligned} \frac{dx}{dt} &= \alpha\, x - \beta\, xy \\ \frac{dy}{dt} &= \delta\, xy - \gamma\, y \end{aligned} \tag{78}$$

где:

- $x(t)$ — численность жертв,

- $y(t)$ — численность хищников,

- Параметры: $\alpha = 1.1, \beta = 0.4, \delta = 0.1, \gamma = 0.4.$

Код решения:

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# Определение системы ОДУ
def predator_prey(t, z, alpha, beta, delta, gamma):
    x, y = z
    dxdt = alpha * x - beta * x * y
    dydt = delta * x * y - gamma * y
    return [dxdt, dydt]

# Параметры модели
params = (1.1, 0.4, 0.1, 0.4)  # (α, β, δ, γ)

# Начальные условия: x(0)=10, y(0)=5
initial_conditions = [10, 5]

# Временной интервал: [0, 50] с шагом 0.1
t_span = (0, 50)
t_eval = np.linspace(0, 50, 500)

# Численное решение
solution = solve_ivp(
    fun=predator_prey,
    t_span=t_span,
    y0=initial_conditions,
    t_eval=t_eval,
    args=params
)

# Извлечение результатов
x = solution.y[0]  # траектория жертв
y = solution.y[1]  # траектория хищников
times = solution.t   # временные точки

# Визуализация
plt.figure(figsize=(12, 6))

# График динамики популяций
plt.subplot(1, 2, 1)
plt.plot(times, x, 'g-', label='Жертвы (x)')
plt.plot(times, y, 'r-', label='Хищники (y)')
plt.xlabel('Время')
```

```
plt.ylabel('Численность')
plt.title('Динамика популяций')
plt.grid(True)
plt.legend()

# Фазовый портрет
plt.subplot(1, 2, 2)
plt.plot(x, y, 'b-')
plt.xlabel('Жертвы (x)')
plt.ylabel('Хищники (y)')
plt.title('Фазовый портрет системы')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Пояснения к коду:

1. Определение системы:

Функция predator_prey возвращает правые части уравнений. Аргумент z содержит текущие значения $[x, y]$.

2. Интегратор solve_ivp:

- fun: функция с системой ОДУ
- t_span: временной интервал
- y0: начальные условия
- t_eval: точки, в которых вычисляется решение
- args: дополнительные параметры модели

3. Результаты:

- solution.y[0] — значения $x(t)$
- solution.y[1] — значения $y(t)$
- solution.t — временные точки

4. Визуализация:

- Левый график: динамика численности со временем
- Правый график: фазовый портрет системы (зависимость y от x)

Результаты выполнения (рис. 12):

1. График динамики популяций:

- Показывает циклические колебания численности

- Пик хищников запаздывает относительно пика жертв

2. Фазовый портрет:

- Замкнутая кривая = периодический режим

- Отсутствие сходимости к точке = отсутствие равновесия

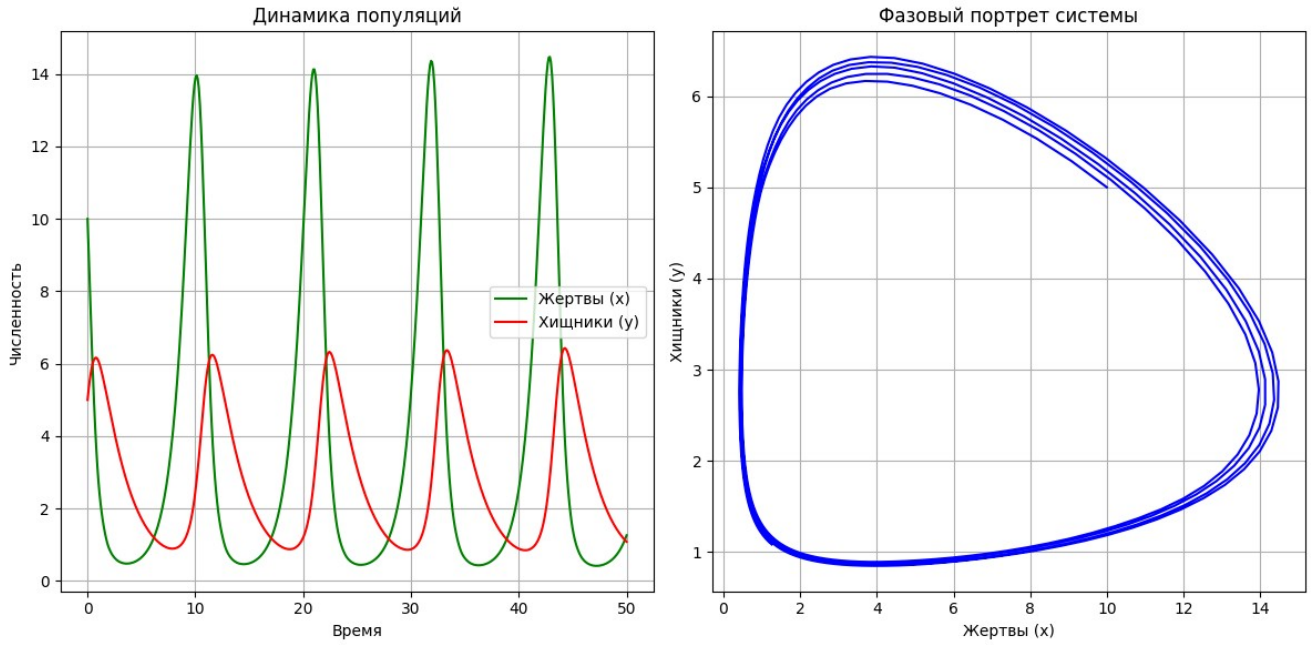


Рисунок 12 - Результаты вычислений по модели "хищник-жертва"

Интерпретация:

Система демонстрирует устойчивые колебания. При росте численности жертв увеличивается популяция хищников, что затем приводит к сокращению жертв и последующему падению численности хищников. Цикл повторяется.

Динамическая модель сложной химической реакции

Рассмотрим сложную химическую реакцию вида:

$$A + B \rightleftharpoons C; \quad (79)$$

$$C + D \rightarrow E; \quad (80)$$

$$E \rightarrow F \quad (81)$$

Реакция (79) обратимая. Обозначим константы скорости прямой и обратной реакций посредством k_1 и k_r, соответственно. Константы скоростей реакций (80),(81) обозначим посредством k_2 и k_3, соответственно.

Рассмотрим кинетику реакции с использованием закона действующих масс. Согласно закону действующих масс скорость изменения концентрации реагента в простой химической реакции равна произведению константы скорости на концентрации реагирующих веществ со знаком «плюс» если данное вещество в этой реакции образуется и со знаком «минус», если вещество расходуется. В сложной реакции скорость изменения концентрации реагента равна алгебраической сумме скоростей всех реакций, в которых данный реагент участвует. Таким образом для реакций (79)-(81) скорости изменения концентраций всех участвующих в реакции веществ описывается следующей системой дифференциальными дифференциальных уравнений:

$$\begin{aligned}
\frac{d[A]}{dt} &= -k_1[A][B] + k_r[C] \\
\frac{d[B]}{dt} &= -k_1[A][B] + k_r[C] \\
\frac{d[C]}{dt} &= k_1[A][B] - k_r[C] - k_2[C][D] \\
\frac{d[D]}{dt} &= -k_2[C][D] \\
\frac{d[E]}{dt} &= k_2[C][D] - k_3[E] \\
\frac{d[F]}{dt} &= k_3[E]
\end{aligned} \tag{82}$$

где:

- k_1 = 0.05 (константа скорости прямой реакции A+B → C)
- k_r = 0.02 (константа скорости обратной реакции C → A+B)
- k_2 = 0.1 (константа скорости реакции C+D → E)
- k_3 = 0.15 (константа скорости реакции E → F)

Код для численного решения:

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

# Параметры реакций
```

```
k1 = 0.05  # A+B -> C
kr = 0.02  # C -> A+B
k2 = 0.1   # C+D -> E
k3 = 0.15  # E -> F

# Функция системы ОДУ
def chemical_kinetics(t, conc):
    A, B, C, D, E, F = conc

    # Скорости реакций
    r1 = k1 * A * B    # A+B -> C
    rr = kr * C        # C -> A+B
    r2 = k2 * C * D    # C+D -> E
    r3 = k3 * E        # E -> F

    # Система уравнений
    dA_dt = -r1 + rr
    dB_dt = -r1 + rr
    dC_dt = r1 - rr - r2
    dD_dt = -r2
    dE_dt = r2 - r3
    dF_dt = r3

    return [dA_dt, dB_dt, dC_dt, dD_dt, dE_dt, dF_dt]

# Начальные концентрации
initial_conc = [
    2.0,  # [A]0
    1.5,  # [B]0
    0.0,  # [C]0
    1.2,  # [D]0
    0.0,  # [E]0
    0.0   # [F]0
]

# Временной интервал
t_span = (0, 100)
t_eval = np.linspace(0, 100, 1000)

# Решение системы
solution = solve_ivp(
    fun=chemical_kinetics,
    t_span=t_span,
    y0=initial_conc,
    t_eval=t_eval,
    method='BDF',  # Метод для жестких систем
    rtol=1e-6,
    atol=1e-9
)
```

```
# Извлечение результатов
t = solution.t
A = solution.y[0]
B = solution.y[1]
C = solution.y[2]
D = solution.y[3]
E = solution.y[4]
F = solution.y[5]

# Визуализация
plt.figure(figsize=(14, 10))
plt.suptitle("Кинетика сложной химической реакциинA+B ⇌ C; C+D → E; E → F",
fontsize=16)

# График концентраций
plt.subplot(2, 1, 1)
plt.plot(t, A, 'b-', linewidth=2, label='[A]')
plt.plot(t, B, 'c--', linewidth=2, label='[B]')
plt.plot(t, C, 'g-', linewidth=2, label='[C]')
plt.plot(t, D, 'm-.', linewidth=2, label='[D]')
plt.plot(t, E, 'r-', linewidth=2, label='[E]')
plt.plot(t, F, 'k-', linewidth=3, label='[F]')

plt.ylabel('Концентрация, М')
plt.xlabel('Время, с')
plt.title('Динамика концентраций реагентов')
plt.legend(loc='upper right')
plt.grid(True, alpha=0.3)
plt.gca().xaxis.set_major_locator(MaxNLocator(10))

# График скорости образования продуктов
plt.subplot(2, 1, 2)
plt.plot(t, np.gradient(F, t), 'k-', linewidth=2, label='d[F]/dt')
plt.plot(t, np.gradient(E, t), 'r--', linewidth=1.5, label='d[E]/dt')
plt.plot(t, np.gradient(C, t), 'g-.', linewidth=1.5, label='d[C]/dt')

plt.ylabel('Скорость, М/с')
plt.xlabel('Время, с')
plt.title('Скорость образования продуктов')
plt.legend(loc='upper right')
plt.grid(True, alpha=0.3)
plt.gca().xaxis.set_major_locator(MaxNLocator(10))

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Результаты расчета представлены на рисунке 13.

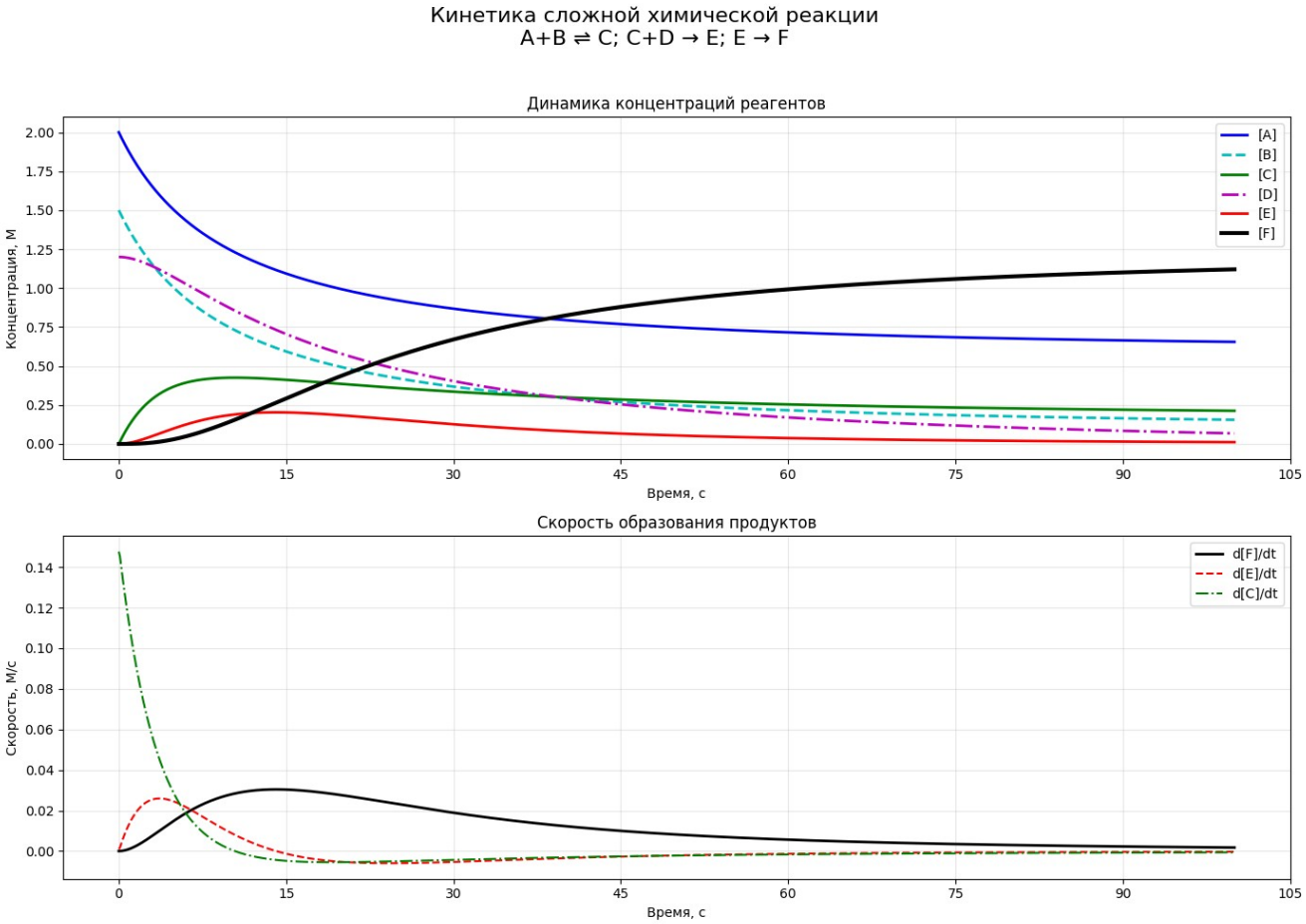


Рисунок 13 - Результаты расчета для модели сложной химической реакции

Ключевые особенности модели:

1. Обратимая первая стадия:
   - Реакция A+B ⇌ C имеет константы прямой и обратной реакций
   - В начальный момент наблюдается быстрое образование C
   - Затем устанавливается квазиравновесие
2. Промежуточные продукты:
   - [C] достигает максимума при t≈12 с
   - [E] накапливается медленнее с максимумом при t≈14 с
   - Концентрация [D] монотонно уменьшается
3. Конечный продукт:
   - [F] плавно нарастает после лаг-периода
   - Скорость образования d[F]/dt имеет максимум при t≈15 с

4. Кинетические особенности:

- Стадия $C+D \rightarrow E$ лимитирует общую скорость процесса
- После исчерпания [D] (t>50 с) система переходит в стадию распада $E \rightarrow F$
- Установление равновесия $A+B \rightleftharpoons C$ наблюдается при t>70 с

Физическая интерпретация:

1. Начальная фаза (0-15 с):
   - Быстрое образование промежуточного продукта C
   - Начало потребления D для образования E
2. Фаза максимальной активности (15-50 с):
   - Основной вклад реакции $C+D \rightarrow E$
   - Накопление E и начало образования F
   - Установление баланса в обратимой реакции
3. Завершающая фаза (50-100 с):
   - Истощение реагента D останавливает вторую стадию
   - Медленный распад E в F
   - Стабилизация обратимой пары $A+B \rightleftharpoons C$
4. Кинетические ограничения:
   - Скорость образования F никогда не превышает скорость распада E
   - Скорость изменения C определяется балансом трех процессов

Практическое применение модели:

1. Оптимизация времени реакции для максимального выхода F
2. Определение лимитирующей стадии процесса
3. Прогнозирование влияния изменения начальных концентраций
4. Анализ чувствительности к константам скорости

Модель демонстрирует характерное поведение для многостадийных химических процессов с обратимыми и необратимыми стадиями, промежуточными продуктами и лимитирующими стадиями.

1.2.2.6. Уравнения в частных производных как динамические модели

Уравнения в частных производных (УрЧП) описывают системы, где состояние зависит от нескольких независимых переменных (пространство + время). Основные классы динамических УрЧП приведены в таблице 3. Производная по времени обозначается индексом t (первая производная - u_t) или $t\,t$ (вторая производная - u_{tt}) у неизвестной переменной $u=u\left(x,y,z,t\right)$.

Таблица 3. Виды уравнений в частных производных

Тип уравнения	Общий вид	Физический смысл	Примеры применения
Параболические	$u_t=a\Delta u$	Диффузия, теплоперенос	Распространение тепла, диффузия газов
Гиперболические	$u_{tt}=c^2\Delta u$	Волновые процессы	Звуковые волны, колебания мембран
Эллиптические	$\Delta u=0$	Стационарные состояния	Равновесная температура, потенциал

Дифференциальные характеристики уравнений:

- Параболические: 1 временная производная (эволюция)
- Гиперболические: 2 временные производные (колебания)
- Эллиптические: нет временных производных (стационарные)

Особенности динамических моделей:

- Параболические и гиперболические описывают эволюцию систем во времени
- Требуют задания начальных и граничных условий
- Пространственные операторы (лапласиан) определяют характер распространения

Параболические уравнения: уравнение теплопроводности

Общий вид:

$$\frac{\partial u}{\partial t}=a\nabla^2 u \tag{83}$$

где:

- $u(\boldsymbol{r}, t)$ - температура в точке $\boldsymbol{r}$ в момент t

- a - коэффициент температуропроводности

- ∇^2 - оператор Лапласа

Физическая интерпретация:

1. Модель диффузии:

   - Поток тепла пропорционален градиенту температуры (закон Фурье:)

   - Скорость изменения температуры зависит от кривизны распределения

Закон теплопроводности Фурье формулирует, что плотность теплового потока, образующегося при теплопроводности, пропорциональна градиенту температуры и направлена в сторону уменьшения температуры. Математически он записывается так:

$$\boldsymbol{q} = -\lambda \nabla T$$

где $\boldsymbol{q}$ — вектор плотности теплового потока (Вт/м²), λ — коэффициент теплопроводности (Вт/(м·К)), ∇T — градиент температуры (К/м).

- Минус указывает, что поток тепла идет в сторону понижения температуры, то есть противоположно направлению роста температуры.

- Коэффициент теплопроводности характеризует способность вещества проводить тепло; его величина зависит от природы, структуры, температуры вещества и других факторов.

- Закон Фурье применим к стационарным (установившимся) процессам и не учитывает инерционность распространения тепла, поэтому не применяется для высокочастотных процессов или условий, когда тепловая волна имеет конечную скорость (например, в случае "второго звука" в криогенных температурах).

2. Эволюция температуры:

```
  # Псевдокод численного решения (метод конечных разностей)
for n in range(time_steps):
    for i in range(1, Nx-1):
        u_new[i] = u[i] + dt*a*(u[i-1] - 2*u[i] + u[i+1])/dx2
```

Характерные свойства:

- Бесконечная скорость распространения возмущений

- Сглаживание начальных неоднородностей

- Монотонное стремление к равновесию

- Сохранение энергии в замкнутой системе

Гиперболические уравнения: волновое уравнение

Общий вид:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \qquad (84)$$

где:

- $u(\boldsymbol{r}, t)$ - амплитуда возмущения

- c - скорость распространения волны

Физическая интерпретация:

1. Модель распространения волн:

  - Восстанавливающая сила пропорциональна кривизне

  - Ускорение элемента среды зависит от разности воздействий

2. Характерные свойства:

  - Конечная скорость распространения (c)

  - Сохранение формы волны (в отсутствие дисперсии)

  - Возможность интерференции и отражения

Численное решение (схема "крест"):

```
# Псевдокод для 1D волнового уравнения
for n in range(1, time_steps):
    for i in range(1, Nx-1):
        u[n+1,i] = 2*u[n,i] - u[n-1,i] + (c*dt/dx)2 * (u[n,i+1] - 2*u[n,i] + u[n,i-1])
```

Постановка задач: начальные и граничные условия

Для корректной формулировки динамической задачи УрЧП необходимо задать:

1. Начальные условия (задают состояние в момент t=0):

- Для параболических уравнений

$$u(\boldsymbol{r}, 0) = u_0(\boldsymbol{r}) \qquad (85)$$

- Для гиперболических уравнений

$$u(\boldsymbol{r},0)=u_0(\boldsymbol{r}),$$
$$\frac{\partial u}{\partial t}(\boldsymbol{r},0)=v_0(\boldsymbol{r}) \quad (86)$$

2. Граничные условия (описывают поведение на границе области). В таблице 4 приведены типы граничных условий для уравнения теплопроводности

Таблица 4. Типы граничных условий для уравнения теплопроводности

Тип условия	Математическая форма	Физический смысл
Дирихле	$T\big\|_{\partial\Omega}=g(\boldsymbol{r},t)$	Фиксированное значение (температура)
Неймана	$\left.\frac{\partial T}{\partial n}\right\|_{\partial\Omega}=h(\boldsymbol{r},t)$	Фиксированный поток (теплоизоляция)
Ньютона-Рихмана (смешанное)	$-\lambda\left.\frac{\partial T}{\partial n}\right\|_{\partial\Omega}=\alpha(T_c-T_{inf})$	Конвективный теплообмен

Теорема существования и единственности решения требует согласованности начальных и граничных условий.

Пример численного решения двумерного уравнения теплопроводности

Рассмотрим нестационарное уравнение теплопроводности для двумерной прямоугольной области

$$\frac{\partial u}{\partial t}=a\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right), 0<x<Lx, 0<y<Ly, t>0. \quad (87)$$

где a — коэффициент температуропроводности

1. Параметры модели:

- Размер пластины: 1x1 м
- Коэффициент температуропроводности: 0.01 м²/с
- Время моделирования: 5 с
- Сетка: 51x51 узел

2. Начальное условие:

- Гауссово распределение температуры в центре пластины (10°C)

- Температура краев: 0°C

3. Граничные условия:

- Левая и нижняя границы: Дирихле (0°C)

- Правая граница: Неймана (теплоизоляция)

- Верхняя граница: Робина (конвективный теплообмен)

4. Численная схема:

- Явная разностная схема (forward in time, central in space)

- Автоматический подбор шага по времени для устойчивости

- Учет смешанных граничных условий

5. Визуализация:

- 3D-анимация распределения температуры

- Тепловая карта в 2D

- Распределение температуры по центральным линиям

- График изменения полной тепловой энергии

Шаги решения:

1. Дискретизация пространства и времени

2. Аппроксимация производных конечными разностями

3. Реализация явной схемы интегрирования

4. Применение граничных условий

5. Визуализация результатов

Ниже приведен текст программы, решающей данную задачу.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from mpl_toolkits.mplot3d import Axes3D

# Параметры задачи
Lx = 1.0       # Длина пластины по x [м]
Ly = 1.0       # Длина пластины по y [м]
alpha = 0.01   # Коэффициент температуропроводности [м²/с]
T = 5.0        # Общее время моделирования [с]

# Параметры сетки
Nx = 51        # Количество узлов по x
Ny = 51        # Количество узлов по y
```

```
Nt = 500       # Количество шагов по времени
dx = Lx / (Nx - 1)
dy = Ly / (Ny - 1)
dt = T / Nt

# Проверка устойчивости
stability = alpha * dt * (1/dx**2 + 1/dy**2)
if stability > 0.5:
    dt = 0.5 / (alpha * (1/dx**2 + 1/dy**2))
    Nt = int(T / dt) + 1
    print(f"Шаг по времени уменьшен до {dt:.4f} с для устойчивости")

# Создание сетки
x = np.linspace(0, Lx, Nx)
y = np.linspace(0, Ly, Ny)
X, Y = np.meshgrid(x, y)

# Начальное условие (гауссов пик в центре)
u0 = np.zeros((Ny, Nx))
cx, cy = Lx/2, Ly/2
u0 = 10 * np.exp(-((X - cx)**2 / 0.05 + (Y - cy)**2 / 0.05))

# Граничные условия (смешанные)
def apply_boundary_conditions(u, t):
    # Левая граница: Дирихле (фиксированная температура)
    u[:, 0] = 0.0

    # Правая граница: Неймана (теплоизоляция)
    u[:, -1] = u[:, -2]

    # Нижняя граница: Дирихле
    u[0, :] = 0.0

    # Верхняя граница: Ньютона-Рихмана (конвективный теплообмен)
    h = 5.0  # Коэффициент теплообмена
    u_ambient = 0.0  # Температура окружающей среды
    u[-1, :] = (u[-2, :] + h * dy * u_ambient) / (1 + h * dy)

    # Угловые точки (среднее значение соседей)
    u[0, 0] = (u[0, 1] + u[1, 0]) / 2
    u[0, -1] = (u[0, -2] + u[1, -1]) / 2
    u[-1, 0] = (u[-1, 1] + u[-2, 0]) / 2
    u[-1, -1] = (u[-1, -2] + u[-2, -1]) / 2

    return u

# Применение начальных граничных условий
u = u0.copy()
u = apply_boundary_conditions(u, 0)
```

```
# Массив для хранения решения
u_solution = np.zeros((Nt, Ny, Nx))
u_solution[0] = u

# Основной цикл (явная схема)
for n in range(1, Nt):
    un = u_solution[n-1].copy()

    # Вычисление нового временного слоя
    u[1:-1, 1:-1] = un[1:-1, 1:-1] + alpha * dt * (
        (un[1:-1, 2:] - 2*un[1:-1, 1:-1] + un[1:-1, :-2]) / dx**2 +
        (un[2:, 1:-1] - 2*un[1:-1, 1:-1] + un[:-2, 1:-1]) / dy**2
    )

    # Применение граничных условий
    u = apply_boundary_conditions(u, n*dt)
    u_solution[n] = u.copy()

# Визуализация результатов
fig = plt.figure(figsize=(15, 10))
fig.suptitle('Распределение температуры в пластине', fontsize=16)

# 1. 3D-анимация распределения температуры
ax1 = fig.add_subplot(221, projection='3d')
surf = ax1.plot_surface(X, Y, u_solution[0], cmap='hot', rstride=1, cstride=1)
ax1.set_xlabel('x [м]')
ax1.set_ylabel('y [м]')
ax1.set_zlabel('Температура [°C]')
ax1.set_zlim(0, 10)
ax1.set_title(f'Время: {0:.2f} с')

# 2. Тепловая карта
ax2 = fig.add_subplot(222)
im = ax2.imshow(u_solution[0], cmap='hot', origin='lower',
                extent=[0, Lx, 0, Ly], vmin=0, vmax=10)
plt.colorbar(im, ax=ax2, label='Температура [°C]')
ax2.set_xlabel('x [м]')
ax2.set_ylabel('y [м]')
ax2.set_title('Тепловая карта')

# 3. График температуры по центральным линиям
ax3 = fig.add_subplot(212)
line_x, = ax3.plot(x, u_solution[0, Ny//2, :], 'r-', label='По x (y=Ly/2)')
line_y, = ax3.plot(y, u_solution[0, :, Nx//2], 'b-', label='По y (x=Lx/2)')
ax3.set_xlabel('Координата [м]')
ax3.set_ylabel('Температура [°C]')
ax3.set_ylim(0, 10)
ax3.grid(True)
```

```
ax3.legend()
ax3.set_title('Распределение температуры по осям')

plt.tight_layout(rect=[0, 0, 1, 0.95])

# Функция обновления анимации
def update(frame):
    # Обновление 3D графика
    ax1.clear()
    surf = ax1.plot_surface(X, Y, u_solution[frame], cmap='hot', rstride=1, cstride=1)
    ax1.set_xlabel('x [м]')
    ax1.set_ylabel('y [м]')
    ax1.set_zlabel('Температура [°C]')
    ax1.set_zlim(0, 10)
    ax1.set_title(f'Время: {frame*dt:.2f} с')

    # Обновление тепловой карты
    im.set_data(u_solution[frame])

    # Обновление графиков по осям
    line_x.set_ydata(u_solution[frame, Ny//2, :])
    line_y.set_ydata(u_solution[frame, :, Nx//2])

    return surf, im, line_x, line_y

# Создание анимации
ani = FuncAnimation(fig, update, frames=range(0, Nt, max(1, Nt//100)),
                    interval=50, blit=False)

plt.tight_layout()
plt.show()

# Анализ сохранения энергии
total_energy = np.array([np.sum(frame) * dx * dy for frame in u_solution])
time_points = np.linspace(0, T, Nt)

plt.figure(figsize=(10, 6))
plt.plot(time_points, total_energy, 'b-')
plt.xlabel('Время, с')
plt.ylabel('Полная тепловая энергия')
plt.title('Эволюция полной тепловой энергии в системе')
plt.grid(True)
plt.show()
```

Программа производит вычисление распределения температуры по площади пластины в различные моменты времени методом конечных разностей. По результатам расчета строится анимированный график распределения температуры в процессе решения уравнения. На рисунке 14 приведены графики распределений в фиксированный момент времени.

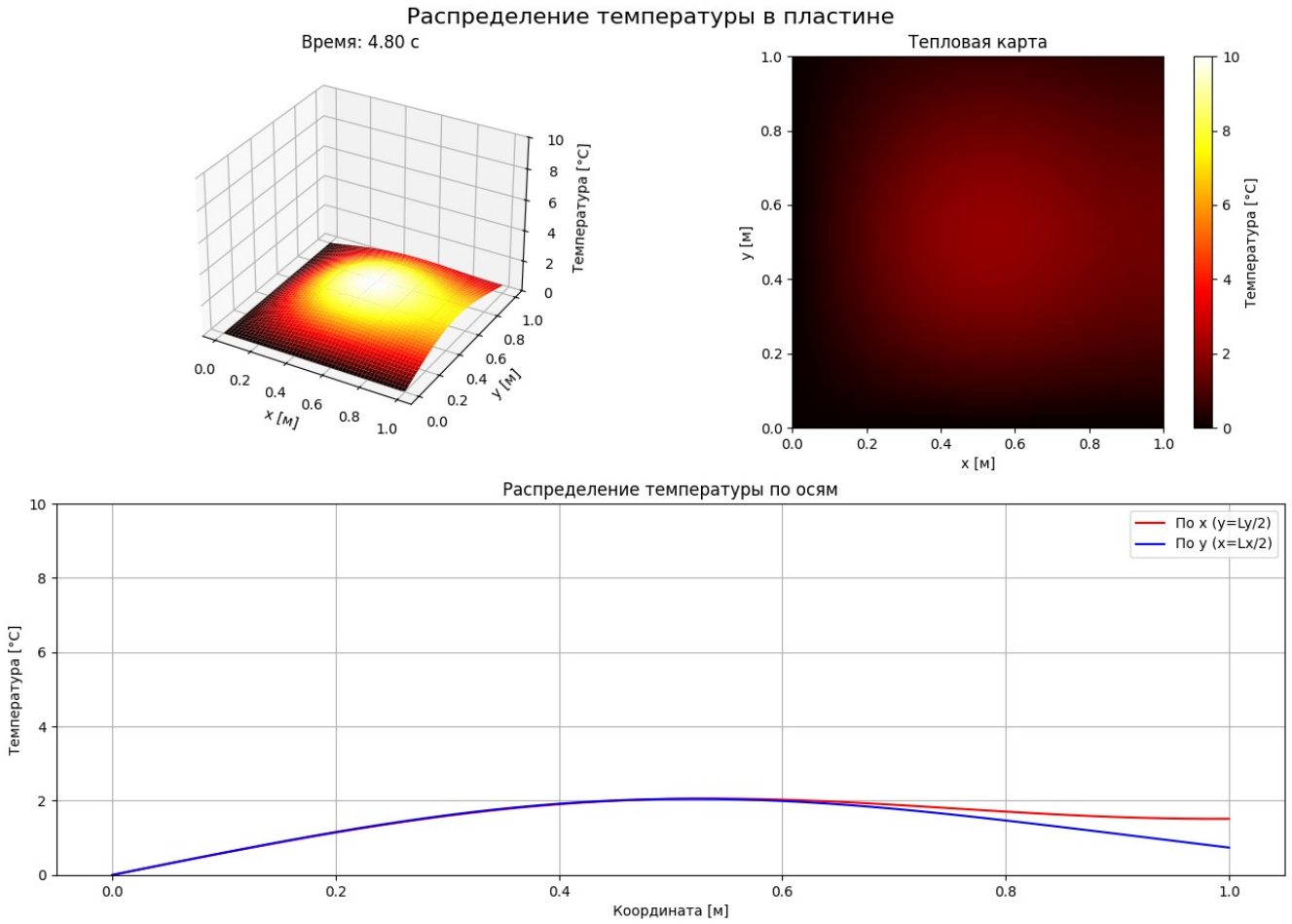


Рисунок 14 - Графики распределения температуры по площади пластины в фиксированный момент времени (4.8 с)

Также строится график изменения полной тепловой энергии в системе в процессе решения (рис), из которого видно что происходит остывание пластины.

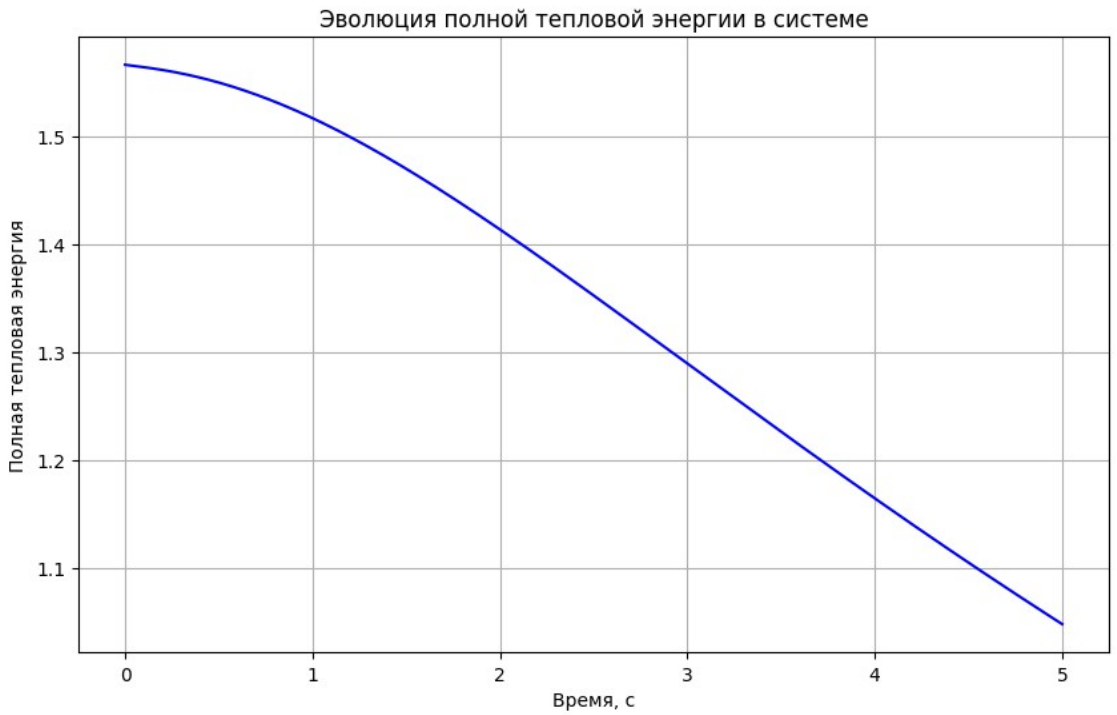


Рисунок 15 - График изменения полной тепловой энергии в пластине

Физическая интерпретация результатов:

1. Начальный момент:
   - Максимум температуры в центре пластины (10°C)
   - Температура плавно убывает к краям
2. Динамика процесса:
   - Быстрое охлаждение центральной области
   - Постепенное выравнивание температуры
   - Особенности у границ с разными условиями
3. Установившийся режим:
   - Распределение температуры определяется граничными условиями
   - Вертикальный градиент на верхней границе (условие Ньютона-Рихмана)
   - Горизонтальный градиент из-за разных условий на левой и правой границах

4. Сохранение энергии:

- Монотонное уменьшение полной энергии

- Скорость охлаждения уменьшается со временем

- Стабилизация при достижении равновесного состояния

Программа демонстрирует ключевые аспекты моделирования теплопереноса в двумерных объектах с использованием явной разностной схемы и визуализации результатов.

2. Стохастические модели систем

В окружающем нас мире практически не существует систем, полностью свободных от случайности. Стохастические модели возникают как ответ на фундаментальную проблему: необходимость описания и анализа систем, подверженных влиянию неконтролируемых факторов.

1. Неопределенность в системах: природа и источники

Неопределенность — неотъемлемая характеристика реальных систем. Её основные источники:

- Внешние возмущения: Случайные воздействия среды (изменение спроса на товар, колебания температуры, помехи в канале связи).

- Внутренняя стохастичность: Вариабельность параметров системы (время обработки детали на станке, длительность химической реакции).

- Неполнота информации: Ограниченность знаний о структуре системы или её поведении (например, неточность измерений).

- Человеческий фактор: Непредсказуемость решений агентов в социально-экономических системах.

Стохастичность — не "погрешность", а фундаментальное свойство таких систем, требующее специальных математических методов.

2. Области применения: где незаменимы стохастические модели

Эти модели критически важны в областях, где случайность определяет поведение системы:

- Технические системы:

- Оценка надёжности сетей связи и энергосистем.

- Оптимизация пропускной способности транспортных узлов.

- Экономика и управление:

- Моделирование финансовых рисков (цены акций, волатильность рынков).

- Управление запасами при случайном спросе.

- Биология и медицина:

  - Распространение эпидемий.

  - Динамика биохимических реакций в клетках.

- Социальные и экологические системы:

  - Прогнозирование пассажиропотоков.

  - Моделирование климатических изменений с учётом случайных факторов.

3. Основные отличия от детерминированных моделей (табл. 5).

Таблица 5. Основные различия детерминированных и стохастических моделей

Критерий	Детерминированные модели	Стохастические модели
Природа параметров	Жёстко заданные константы	Случайные величины/процессы
Результат	Единственный сценарий	Распределение возможных исходов
Анализ	Точные решения уравнений	Вероятностные характеристики (матожидание, дисперсия, квантили)
Устойчивость	Чувствительность к входным данным	Оценка рисков и вариабельности

Детерминированные модели отвечают на вопрос "Что будет?", стохастические — "Что может произойти и с какой вероятностью?".

4. Ключевые задачи стохастического моделирования

- Прогнозирование вероятностных характеристик:

  - Расчёт вероятности отказа технической системы.

  - Оценка среднего времени ожидания в очереди.

- Оптимизация в условиях неопределённости:

  - Выбор стратегии управления запасами при случайном спросе.

  - Определение оптимального числа серверов в call-центре.

- Анализ устойчивости и рисков:

- Построение доверительных интервалов для выходных параметров.

- Оценка Value-at-Risk (VaR) в финансовых моделях.

- Верификация моделей:

- Сравнение теоретических распределений с эмпирическими данными.

- Проверка гипотез о свойствах случайных процессов.

Таким образом, стохастическое моделирование не просто дополняет детерминированный подход — оно предоставляет язык для описания принципиально иного класса явлений. Понимание методов работы со случайностью становится ключевой компетенцией для инженеров, экономистов, биологов и исследователей в эпоху сложных систем.

2.1. Основные понятия теории вероятностей и случайных процессов

2.1.1. Случайные величины: распределения и числовые характеристики

Случайная величина (СВ) — ключевой объект стохастического моделирования, численно описывающий исход случайного эксперимента. Она ставит в соответствие каждому элементарному исходу вещественное число. В зависимости от типа множества значений СВ делятся на:

- Дискретные (принимают конечное или счётное число значений)

- Непрерывные (принимают несчётное множество значений, например, интервал)

Закон распределения задаёт вероятности значений СВ. Некоторые распространенные виды дискретных и непрерывных распределений приведены в таблицах 6,7.

Таблица 6. Дискретные распределения

Распределение	Обозначение	Параметры	Формула вероятности / Плотности	Пример применения
Бернулли	$Bern(p)$	$p \in [0,1]$	$P(X=k)=p^k(1-p)^{1-k}$, $k \in 0,1$	Результат испытания (успех/неудача)
Биномиальное	$Bin(n,p)$	$n \in \mathbb{N}, p \in [0,1]$	$P(X=k)=$ $C_n{}^k\, p^k(1-p)^{n-k}$, $k=0,1,\ldots,n$	Число успехов в n испытаниях
Пуассона	$Pois(\lambda)$	$\lambda>0$ (интенсивность)	$P(X=k)=(\lambda^k e^{-\gamma})/k!$, k=0,1,2,...	Число редких событий за фиксированное время

Ниже приведен текст программы, иллюстрирующей графики дискретных распределений:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import bernoulli, binom, poisson

# Настройка стиля графиков
plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(15, 5))

# 1. Распределение Бернулли (p = 0.7)
plt.subplot(1, 3, 1)
p_bernoulli = 0.7
x_bernoulli = [0, 1]
pmf_bernoulli = bernoulli.pmf(x_bernoulli, p_bernoulli)
plt.bar(x_bernoulli, pmf_bernoulli, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Распределение Бернулли (p=0.7)', fontsize=12)
plt.xlabel('Значения', fontsize=10)
plt.ylabel('Вероятность', fontsize=10)
plt.xticks([0, 1])
plt.ylim(0, 1)

# 2. Биномиальное распределение (n=10, p=0.5)
plt.subplot(1, 3, 2)
n_binom, p_binom = 10, 0.5
x_binom = np.arange(0, n_binom + 1)
```

```
pmf_binom = binom.pmf(x_binom, n_binom, p_binom)
plt.bar(x_binom, pmf_binom, color='lightgreen', edgecolor='black', alpha=0.7)
plt.title('Биномиальное распределение (n=10, p=0.5)', fontsize=12)
plt.xlabel('Число успехов', fontsize=10)
plt.ylabel('Вероятность', fontsize=10)
plt.xticks(x_binom)

# 3. Распределение Пуассона (λ=3)
plt.subplot(1, 3, 3)
lambda_poisson = 3
x_poisson = np.arange(0, 11)  # От 0 до 10 включительно
pmf_poisson = poisson.pmf(x_poisson, lambda_poisson)
plt.bar(x_poisson, pmf_poisson, color='salmon', edgecolor='black', alpha=0.7)
plt.title('Распределение Пуассона (λ=3)', fontsize=12)
plt.xlabel('Число событий', fontsize=10)
plt.ylabel('Вероятность', fontsize=10)
plt.xticks(x_poisson)

# Настройка общего макета и вывод графиков
plt.tight_layout()
plt.show()
```

Программа использует библиотеки:

numpy для работы с числовыми массивами,

matplotlib.pyplot для визуализации,

scipy.stats для статистических функций.

Распределение Бернулли:

Параметр: p = 0.7 (вероятность успеха),

Значения: 0 (неудача) и 1 (успех),

Вероятности рассчитываются через bernoulli.pmf().

Биномиальное распределение:

Параметры: n = 10 (число испытаний), p = 0.5 (вероятность успеха),

Диапазон значений: от 0 до 10,

Вероятности рассчитываются через binom.pmf().

Распределение Пуассона:

Параметр: λ = 3 (среднее количество событий),

Диапазон значений: от 0 до 10 (охватывает основные вероятности),

Вероятности рассчитываются через poisson.pmf().

Программа создает фигуру с тремя графиками в строку (рис. 16):

Левый график: Бернулли с двумя столбцами (0 и 1),

Средний график: Биномиальное распределение (симметричная форма при p=0.5),

Правый график: Пуассона (пик вероятности около λ=3).

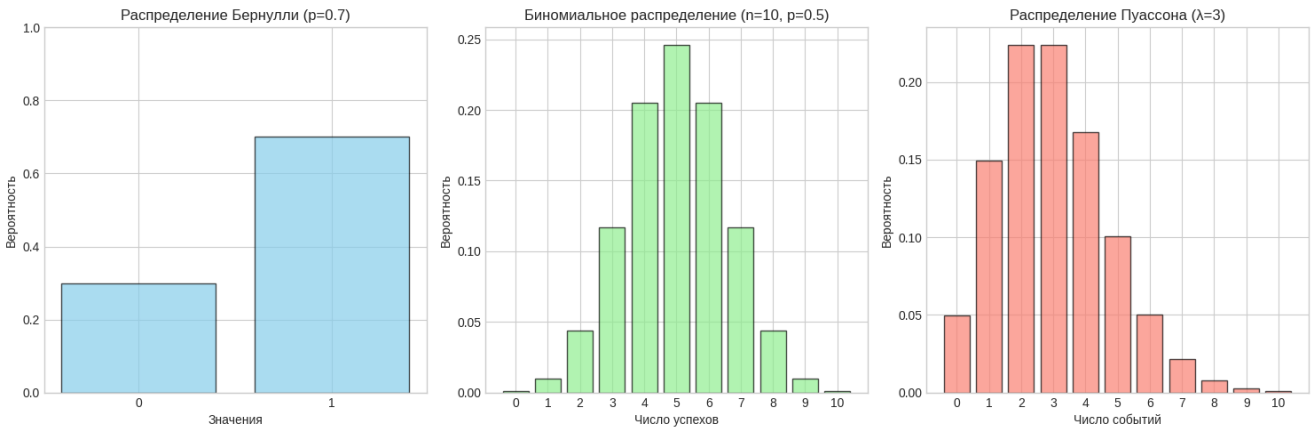


Рисунок 16 - Графики дискретных распределений

Функции распределения случайных величин

Интегральная функция распределения (часто обозначается $F(x)$) — это функция, определяющая для любого числа x вероятность того, что случайная величина X примет значение, меньшее x:

$$F(x)=P(X<x) \tag{88}$$

- Эта функция отражает всю совокупность вероятностных свойств случайной величины и универсальна: применяется для дискретных и непрерывных случайных величин.

- Основные свойства:

- $F(x)$ — неубывающая функция: большему x соответствует не меньшая вероятность:

$$F(x_2)\geq F(x_1) при\, x_2\geq x_1.$$

- Значения $F(x)$ всегда лежат в диапазоне от 0 до 1.

- При $x \to -\infty, F(x) \to 0$; при $x \to +\infty, F(x) \to 1$.

- Для дискретных СВ F(x) — ступенчатая функция с разрывами, а для непрерывных — гладкая и непрерывно дифференцируемая.

- Вероятность попасть в промежуток (a, b) можно выразить через интегральную функцию как разность:

$$P(a<X<b)=F(b)-F(a) \tag{89}$$

Дифференциальная функция распределения, или плотность вероятности (обозначается $f(x)$), — это производная от интегральной функции распределения:

$$f(x)=\frac{dF(x)}{dx} \tag{90}$$

Смысл: вероятность того, что X попадёт в бесконечно малый интервал около x, равна $f(x)\,dx$.

Для любого конечного интервала вероятность определяется интегралом:

$$P(a<X<b)=\int_a^b f(x),dx \tag{91}$$

Свойства дифференциальной функции:

- $f(x)\geq 0$для всех x.

- Интеграл $f(x)$ по всей области определения случайной величины равен 1 (нормировка).

- Плотность вероятности не определена для дискретных случайных величин — только для непрерывных.

Связь между ними

- Интегральная функция распределения F(x) — это "накопленная" вероятность до точки x, считается интегралом от плотности вероятности:

$$F(x)=\int_{-\infty}^{x} f(t),dt \tag{92}$$

- Дифференциальная функция $f(x)$ — это мгновенная скорость приращения вероятности в точке x для непрерывной случайной величины: $f(x)=F'(x)$

- Для дискретных случайных величин плотность не используется, а функция распределения имеет скачки.

В таблице 7 приведены функции плотности экспоненциального и нормального распределений.

Таблица 7. Непрерывные распределения

Распределение	Обозначение	Параметры	Функция плотности f(x)	Пример применения
Экспоненциальное	Exp(λ)	λ > 0 (интенсивность)	$f(x)=\lambda e^{-\varphi x}$, x ≥ 0	Время между событиями (отказы, вызовы)
Нормальное	N(μ,σ²)	μ∈ℝ, σ>0	$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-m)^2}{2\sigma^2}}$	Погрешности измерений, рост популяции

- Экспоненциальное распределение не имеет памяти: $P(X>t+s|X>s)=P(X>t)$.

- Нормальное распределение универсально благодаря Центральной Предельной Теореме

Ниже приведена программа для построения графиков функций экспоненциального и нормального распределений.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import expon, norm

# Настройка стиля и размера графиков
plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(12, 10))

# Параметры распределений
lambda_exp = 0.5  # Параметр экспоненциального распределения (λ)
mu_norm = 0       # Среднее нормального распределения (μ)
sigma_norm = 1    # Стандартное отклонение (σ)

# Создание диапазона значений x
x_exp = np.linspace(0, 10, 500)  # Экспоненциальное: от 0 до 10
x_norm = np.linspace(-5, 5, 500) # Нормальное: от -5 до 5

# 1. Экспоненциальное распределение
# PDF (плотность вероятности)
```

```
pdf_exp = expon.pdf(x_exp, scale=1/lambda_exp)  # scale = 1/λ
# CDF (функция распределения)
cdf_exp = expon.cdf(x_exp, scale=1/lambda_exp)

# 2. Нормальное распределение
# PDF (плотность вероятности)
pdf_norm = norm.pdf(x_norm, mu_norm, sigma_norm)
# CDF (функция распределения)
cdf_norm = norm.cdf(x_norm, mu_norm, sigma_norm)

# Построение графиков
# Экспоненциальное распределение: PDF и CDF
plt.subplot(2, 2, 1)
plt.plot(x_exp, pdf_exp, 'b-', linewidth=2)
plt.title('Экспоненциальное распределение (λ=0.5)\nПлотность вероятности (PDF)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(x_exp, cdf_exp, 'r-', linewidth=2)
plt.title('Экспоненциальное распределение (λ=0.5)\nФункция распределения (CDF)')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.grid(True)

# Нормальное распределение: PDF и CDF
plt.subplot(2, 2, 3)
plt.plot(x_norm, pdf_norm, 'b-', linewidth=2)
plt.title(f'Нормальное распределение (μ=0, σ=1)\nПлотность вероятности (PDF)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)

plt.subplot(2, 2, 4)
plt.plot(x_norm, cdf_norm, 'r-', linewidth=2)
plt.title(f'Нормальное распределение (μ=0, σ=1)\nФункция распределения (CDF)')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.grid(True)

# Настройка компоновки и отображение
plt.tight_layout()
plt.show()
```

Программа построит 4 графика (рис. 17):

- Экспоненциальное PDF: убывающая экспонента

- Экспоненциальное CDF: возрастающая кривая к 1
- Нормальное PDF: "колокол" Гаусса
- Нормальное CDF: S-образная сигмоида

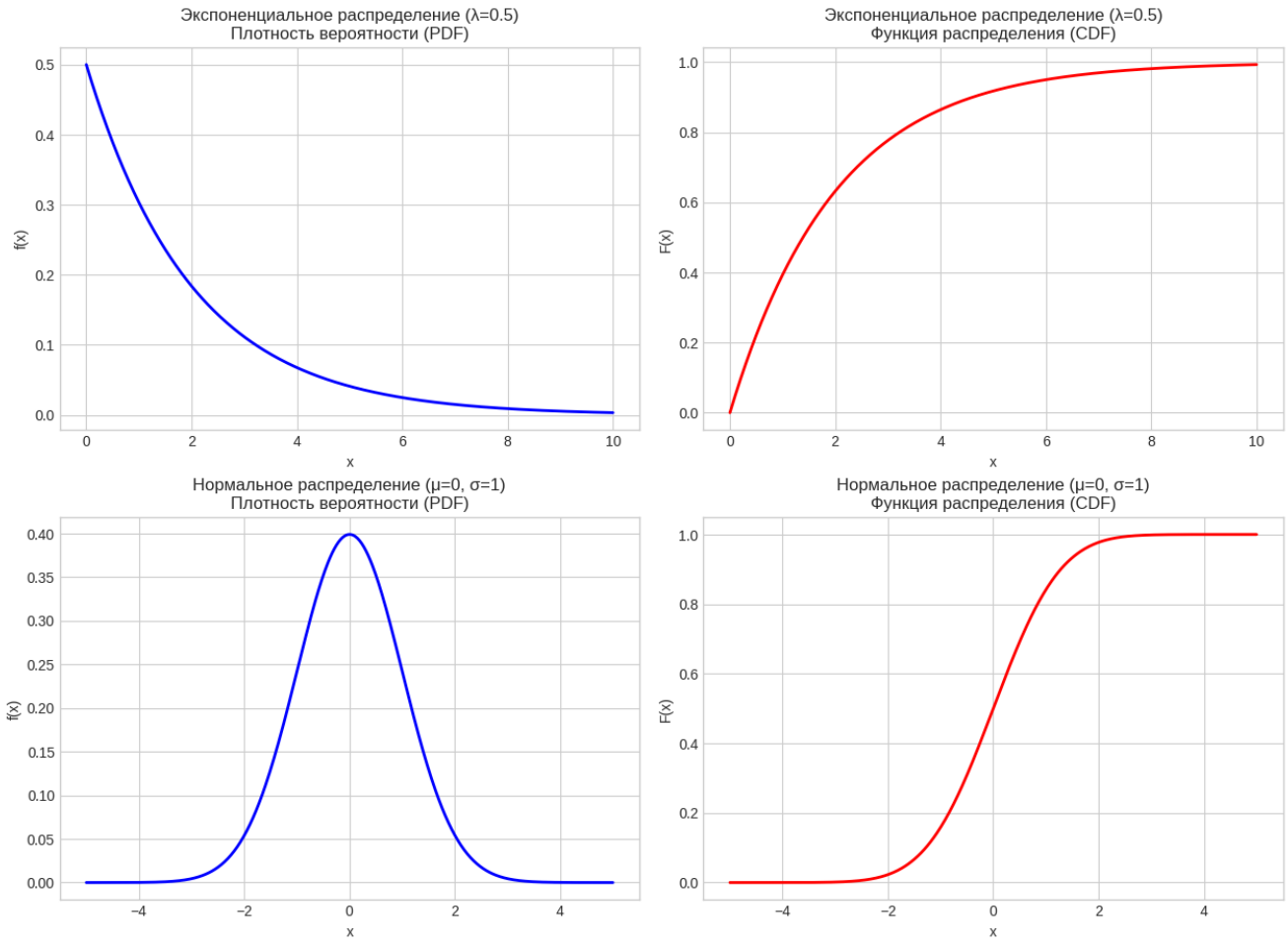


Рисунок 17 - Графики функций экспоненциального и нормального распределений

Числовые характеристики случайных величин

Эти величины позволяют описать СВ компактно, без знания всего распределения.

Основные характеристики

1. Математическое ожидание (МО) — "центр тяжести" распределения:

- Дискретная СВ: $E[X]=\Sigma\, x_i \cdot p_i$

- Непрерывная СВ: $E[X]=\int x \cdot f(x)\,dx$

Примеры: $E[Bin(n,p)]=n \cdot p$, $E[\exp(\lambda)]=1/\lambda$.

2. Дисперсия — мера разброса значений вокруг МО:

$Var(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2$

Примеры: $Var(Bin(n, p)) = n \cdot p \cdot (1 - p)$, $Var(N(\mu, \sigma^2)) = \sigma^2$.

3. Стандартное отклонение

$\sigma(X) = \sqrt{Var(X)}$ (интерпретируется в единицах измерения СВ).

Моменты распределения

- Начальный момент порядка k: $\nu_k = E[X^k]$

- Центральный момент порядка k: $\mu_k = E[(X - E[X])^k]$

Частные случаи:

- $\mu_1 = 0$

- $\mu_2 = Var(X)$

- μ_3 характеризует асимметрию (скошенность) распределения.

- μ_4 характеризует островершинность (эксцесс).

Дополнительные инструменты

- Производящая функция (для дискретных СВ): $G(z) = E[z^x] = \Sigma\, p_k \cdot z^k$

- Характеристическая функция (универсальна): $\varphi(t) = E[e^{itx}]$

Используются для вычисления моментов и анализа свойств распределений.

Значение для моделирования

- Выбор распределения определяет адекватность модели (например, поток заявок → Пуассон, время обслуживания → Экспоненциальное).

- Числовые характеристики позволяют:

  - Сравнивать системы по эффективности ($E[X]$ → средняя производительность).

  - Оценивать риски ($Var(X)$ → устойчивость системы к возмущениям).

- Упрощать расчёты (аппроксимация сложных распределений нормальным).

Оценка параметров функций распределения по экспериментальным данным

Оценка параметров функций распределения по экспериментальным данным — это задача, при которой по выборочным (экспериментальным) данным определяют значения параметров теоретической функции распределения случайной величины. Например: по выборке находят среднее и дисперсию для нормального распределения, или параметр λ для распределения Пуассона.

Основные методы оценки параметров

1. Метод моментов

- Приравниваются выборочные моменты (средние, дисперсии и т.д.) к соответствующим теоретическим моментам распределения, из этого получают формулы для оценки параметров.

- Например, для нормального распределения:

- Оценка среднего: выборочное среднее

- Оценка дисперсии: исправленная выборочная дисперсия.

2. Метод максимального правдоподобия (ММП)

- Строится функция правдоподобия — вероятность получить именно эту выборку при неизвестных параметрах.

- В качестве оценки выбирают значения параметров, при которых эта функция достигает максимума.

- Метод универсален и дает хорошие свойства оценок при больших выборках (несмещённость, состоятельность, эффективность).

3. Метод квантилей

- Оценки определяются приравниванием эмпирических и теоретических квантилей (например, медиан, перцентилей).

- Используется, когда моменты либо вычислять трудно, либо распределение их не определяет однозначно.

Типы оценок

- Точечные оценки — одно числовое значение для параметра (например, среднее или дисперсия).

- Интервальные оценки — диапазон значений, с заданной вероятностью содержащий истинный параметр (например, доверительный интервал).

Свойства «хорошей» оценки

- Несмещённость — математическое ожидание оценки совпадает с истинным значением параметра.

- Состоятельность — при увеличении объема выборки оценка стремится к истинному значению параметра.

- Эффективность — минимальная возможная дисперсия оценки среди всех несмещённых оценок.

Пример

Пусть выборка $x_1, \dots, x_n$ из нормального распределения. Тогда:

- Точечная оценка среднего μ:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{93}$$

- Точечная оценка дисперсии σ^2:

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{94}$$

Для других распределений (например, экспоненциального, пуассоновского), формулы будут другими, но методы аналогичны.

Алгоритм в общем виде

1. Соберите выборку экспериментальных данных.

2. Предположите вид закона распределения.

3. Выберите метод (моментов, максимального правдоподобия, квантилей).

4. Подставьте выборочные значения в соответствующие формулы — получите оценки параметров.

5. (Дополнительно) Проверьте согласие гипотезы о законе распределения с экспериментальными данными с помощью критериев согласия (Пирсона, Колмогорова–Смирнова и др.).

Это, так называемые, точечные оценки параметров. Они позволяют судить о значении параметра, но не говорят о том в каком диапазоне точное значение параметра может быть распределено. Такую задачу решает интервальная оценка параметров.

Интервальная оценка — это диапазон значений, в который с заданной вероятностью попадает неизвестный параметр распределения. Такой диапазон называют доверительным интервалом, а вероятность охвата истинного значения параметра — доверительной вероятностью или уровнем доверия (например, 95% или 99%).

Доверительный интервал для среднего нормального распределения

- Если дисперсия известна:

$$\left(\bar{x} - z_{\alpha/2} \frac{\sigma}{\sqrt{n}}, \quad \bar{x} + z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \right) \tag{95}$$

где

$\bar{x}$ — выборочное среднее,

σ — известное стандартное отклонение,

n — объем выборки,

$z_{\alpha/2}$ — квантиль стандартного нормального распределения для уровня значимости α (например, для 95% доверия $z_{0.025} \approx 1.96$).

- Если дисперсия неизвестна (типично):

$$\left(\bar{x} - t_{\alpha/2,n-1} \frac{s}{\sqrt{n}}, \quad \bar{x} + t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} \right) \tag{96}$$

где

s — выборочное стандартное отклонение, вычисленное по несмещенной оценке дисперсии (94)

$t_{\alpha/2,n-1}$ — квантиль t-распределения Стьюдента с $n-1$ степенями свободы.

Доверительный интервал для дисперсии

- При неизвестном математическом ожидании

$$\left(\frac{(n-1)S^2}{\chi^2_{1-\alpha/2}}, \quad \frac{(n-1)S^2}{\chi^2_{\alpha/2}} \right) \tag{97}$$

где

S^2 — выборочная дисперсия (94),

χ^2_p — квантиль хи-квадрат распределения с $n-1$ степенями свободы.

Что означает доверительная вероятность?

Если мы, например, построили 95% доверительный интервал, то это значит: если бы мы многократно повторяли эксперимент и строили интервалы, то примерно 95% из них накрывали бы истинное значение параметра. Используя интервальные оценки, вы не только получаете приблизительное значение параметра, но и выражаете уровень своей уверенности в нём. Такой подход позволяет использовать экспериментальные данные для количественного описания вероятностных закономерностей исследуемого явления.

Ниже приведен пример программы, которая генерирует выборку нормально распределенной случайной величины с заданными значениями параметров и вычисляет точечные и интервальные оценки параметров на основании выборки.

```
import numpy as np
from scipy import stats

# Заданные параметры распределения
true_mean = 10       # Истинное математическое ожидание
true_std = 2         # Истинное стандартное отклонение
n = 40               # Объем выборки
alpha = 0.05         # уровень значимости для 95% доверительного интервала

# Генерация выборки
np.random.seed(0)    # Для воспроизводимости
data = np.random.normal(loc=true_mean, scale=true_std, size=n)

# Точечные оценки
sample_mean = np.mean(data)
sample_var = np.var(data, ddof=1)

# Доверительный интервал для среднего (дисперсия неизвестна)
t_crit = stats.t.ppf(1 - alpha/2, df=n-1)
mean_se = np.std(data, ddof=1) / np.sqrt(n)
mean_ci = (sample_mean - t_crit * mean_se, sample_mean + t_crit * mean_se)

# Доверительный интервал для дисперсии (через Хи-квадрат)
chi2_lower = stats.chi2.ppf(alpha/2, df=n-1)
chi2_upper = stats.chi2.ppf(1 - alpha/2, df=n-1)
var_ci = ((n-1) * sample_var / chi2_upper, (n-1) * sample_var / chi2_lower)

print(f"Истинное среднее: {true_mean}")
print(f"Точечная оценка среднего: {sample_mean:.3f}")
print(f"95% доверительный интервал для среднего: [{mean_ci[0]:.3f}; {mean_ci[1]:.3f}]")
print()
print(f"Истинная дисперсия: {true_std**2}")
print(f"Точечная оценка дисперсии: {sample_var:.3f}")
print(f"95% доверительный интервал для дисперсии: [{var_ci[0]:.3f}; {var_ci[1]:.3f}]")
```

Пример вывода результатов:

Истинное среднее: 10

Точечная оценка среднего: 10.625

95% доверительный интервал для среднего: [9.936; 11.315]

Истинная дисперсия: 4

Точечная оценка дисперсии: 4.647

95% доверительный интервал для дисперсии: [3.119; 7.662]

Увеличивая объем выборки можно наблюдать, как оценки параметров приближаются к истинным значениям.

Проверка статистических гипотез

Проверка статистических гипотез — это процедура статистического анализа, позволяющая по данным выборки сделать вывод о справедливости предполагаемых свойств генеральной совокупности, например, об истинном значении математического ожидания, дисперсии, форме распределения и т.д.

Ключевые определения

- *Статистическая гипотеза* — предположение о распределении и параметрах случайной величины (например, «среднее равно 10»).

- *Нулевая гипотеза* H_0 — базовое утверждение («никаких эффектов нет», «среднее не изменилось»).

- *Альтернативная гипотеза* H_1 — противоположное утверждение («есть эффект», «среднее изменилось»).

Алгоритм проверки гипотезы

1. Формулируйте H_0 и H_1

Пример: $H_0: \mu = 10$, $H_1: \mu \neq 10$

2. Задайте уровень значимости α

Типично $\alpha = 0.05$ означает: вероятность ошибочного отклонения H_0 — не более 5%.

3. Выберите статистический критерий

- t-критерий, F-критерий, критерий χ^2 и др., в зависимости от задачи.

4. Вычислите статистику по критерию

Пример: для t-критерия — посчитать t-статистику по выборке.

5. Сформулируйте правило принятия решения

Если значение статистики попало в критическую область (за пределами допустимого интервала), отклоняем H_0 ; иначе — не отклоняем.

Ошибки при принятии решения

- Ошибка I рода (α) — отклонить верную H_0 .

- Ошибка II рода (β) — не отклонить ложную H_0 .

- Мощность критерия — вероятность правильно отвергнуть ложную H_0 , равна $1-\beta$.

Пример: одновыборочный t-тест

- Задача: проверить, отличается ли среднее от гипотетического значения %mu_0.

- Формулировка: H_0: %mu = %mu_0 ; H_1: %mu <> %mu_0 .

- Критерий: t-критерий Стьюдента при неизвестной дисперсии.

- Далее: сравнивают полученное значение t-статистики с критическим значением, соответствующим уровню значимости alpha.

Универсальный пятиступенчатый алгоритм

1. Формулировать H_0 и H_1

2. Задать уровень значимости (alpha)

3. Выбрать подходящий статистический критерий

4. Определить область критических значений

5. Принять решение по результатам расчетов и сравнения с критическими значениями

Применение статистических критериев проверки гипотез позволяет делать обоснованные выводы о свойствах исследуемых данных и достоверности наблюдаемых эффектов.

Формулы статистик для проверки распространённых гипотез

Статистика Шапиро–Уилка W гипотезы о нормальном виде выборочного распределения вычисляется по формуле:

$$W = \frac{\left(\sum_{i=1^n} a_i x_{(i)}\right)^2}{\sum_{i=1^n} \left(x_i - \bar{x}\right)^2} \tag{98}$$

где:

- $x_{(i)}$ — i-й по порядку (от самого малого к самому большому) элемент выборки размера n ;

- $\bar{x}=\frac{1}{n}\sum_{i=1}^{n} x_i$ — выборочное среднее;

- a_i — специальные коэффициенты, зависящие от размера выборки, вычисляемые через матожидания и ковариации порядковых статистик стандартного нормального распределения (их обычно берут из таблиц или вычисляют автоматически программой).

Объяснение:

- Числитель отражает согласованность порядка элементов данных с порядком теоретических квантилей нормального распределения.

- Знаменатель — «обычная» сумма квадратов отклонений от среднего (несмещённая оценка дисперсии).

Если W близко к 1 — распределение близко к нормальному.

Кратко о коэффициентах a_i :

Коэффициенты a_i определяются конструкцией:

$$(a_1,\dots,a_n)=\frac{m^T V^{-1}}{C} \tag{99}$$

где:

- m — вектор математических ожиданий порядковых статистик стандартной нормали,

- V — их ковариационная матрица,

- C — норма $V^{-1}m$.

Обычно их вычисляет специализированный софт (SciPy, R, SPSS и др.).

2) Проверка значимости различий дисперсий по критерию Фишера (F-критерий)

- Статистическая гипотеза:

- $H_0:\sigma_1^2=\sigma_2^2$

- $H_1:\sigma_1^2\neq\sigma_2^2$

- Статистика критерия:

$$F=\frac{s_1^2}{s_2^2} \tag{100}$$

где:

- s_1^2, s_2^2 — выборочные дисперсии двух независимых выборок (берём большую в числитель, меньшую — в знаменатель).

- Сравнивают рассчитанное F с критическим значением из таблицы распределения Фишера при необходимом уровне значимости и соответствующих степенях свободы.

3) Проверка значимости различий средних (t-критерий Стьюдента) при условии равенства дисперсий

- Статистическая гипотеза:

- $H_0: \mu_1 = \mu_2$

- $H_1: \mu_1 \neq \mu_2$

- Формула статистики (при равенстве дисперсий):

$$t = \frac{\overline{x_1} - \overline{x_2}}{S_p \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{101}$$

где:

- $\overline{x_1}$, $\overline{x_2}$ — выборочные средние,

- n_1, n_2 — размеры выборок,

- S_p — объединённая (пуллированная) оценка стандартного отклонения:

$$S_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}} \tag{102}$$

- s¿1^2, s_2^2 — выборочные дисперсии.

- Если дисперсии заведомо неодинаковы, используют критерий Уэлча (Welch's t-test):

$$t = \frac{\overline{x_1} - \overline{x_2}}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{103}$$

- Порядок действий:

1. Проверить равенство дисперсий (F-критерий).

2. Если дисперсии равны — стандартный t-критерий. Если нет — t-критерий Уэлча.

Ниже приведен пример программы, иллюстрирующий проверку статистических гипотез

```
import numpy as np
from scipy import stats
```

```
np.random.seed(42)  # для воспроизводимости

# 1. Генерация двух выборок
n1, n2 = 30, 35
mean1, std1 = 10, 2
mean2, std2 = 12, 2.2

data1 = np.random.normal(loc=mean1, scale=std1, size=n1)
data2 = np.random.normal(loc=mean2, scale=std2, size=n2)

# 2. Проверка нормальности (критерий Шапиро–Уилка)
shapiro1 = stats.shapiro(data1)
shapiro2 = stats.shapiro(data2)

print('Проверка нормальности (Шапиро–Уилка):')
print(f'  Выборка 1: W = {shapiro1.statistic:.3f}, p = {shapiro1.pvalue:.4f}')
print(f'  Выборка 2: W = {shapiro2.statistic:.3f}, p = {shapiro2.pvalue:.4f}')
if shapiro1.pvalue > 0.05 and shapiro2.pvalue > 0.05:
    print('  Обе выборки не противоречат нормальности (на уровне 0.05)')
else:
    print('  Хотя бы одна выборка противоречит нормальности!')

print('\nСравнение дисперсий (F-критерий Фишера):')
# 3. Сравнение дисперсий
var1 = np.var(data1, ddof=1)
var2 = np.var(data2, ddof=1)
# Берем большую дисперсию в числитель:
if var1 > var2:
    F = var1 / var2
    dfn, dfd = n1-1, n2-1
else:
    F = var2 / var1
    dfn, dfd = n2-1, n1-1

p_f = 2 * min(stats.f.cdf(F, dfn, dfd), 1 - stats.f.cdf(F, dfn, dfd))  # двусторонний p-value
print(f'  F = {F:.3f}, p (двусторонний) = {p_f:.4f}')
equal_var = p_f > 0.05
print(f'  {"Дисперсии не различаются существенно." if equal_var else "Дисперсии
различаются!"}')

# 4. Сравнение средних (t-критерий)
print('\nСравнение средних (t-критерий):')
t_stat, p_val = stats.ttest_ind(data1, data2, equal_var=equal_var)
print(f'  t = {t_stat:.3f}, p = {p_val:.4f}')
if p_val < 0.05:
    print('  Средние статистически различаются (p < 0.05)')
else:
    print('  Нет статистически значимых различий средних (p >= 0.05)')
```

Программа выполняет следующие этапы:

1) Генерирует две выборки из нормального распределения (можно менять параметры).

2) Проверяет каждую на нормальность (критерий Шапиро–Уилка).

3) Сравнивает дисперсии (F-критерий Фишера).

4) Сравнивает средние значения (t-критерий Стьюдента, с учетом исхода сравнения дисперсий).

Пример вывода результатов:

Проверка нормальности (Шапиро–Уилка):

Выборка 1: W = 0.975, p = 0.6868

Выборка 2: W = 0.984, p = 0.8758

Обе выборки не противоречат нормальности (на уровне 0.05)

Сравнение дисперсий (F-критерий Фишера):

F = 1.239, p (двусторонний) = 0.5597

Дисперсии не различаются существенно.

Сравнение средних (t-критерий):

t = -4.229, p = 0.0001

Средние статистически различаются (p < 0.05)

2.1.2. Введение в случайные процессы: определение и классификация

Случайный процесс (СП) — это математическая модель эволюции системы со случайным поведением во времени. В отличие от случайной величины (которая фиксирует состояние в один момент), СП описывает динамику: $X(t,\omega)$, где:

- $t \in T$ — время (индекс множества),

- $\omega \in \Omega$ — элементарное событие (исход вероятностного пространства).

Простая интерпретация:

- При фиксированном t СП превращается в случайную величину $X(t)$.

- При фиксированном ω СП — это траектория (реализация) процесса $X(t)$ во времени.

Ключевые классификации случайных процессов

Выбор типа модели зависит от природы системы и решаемой задачи.

1. По типу множества состояний S (табл. 8)

Таблица 8. Классификация СП по типу множества состояний

Тип	Множество состояний S	Примеры
Дискретные состояния	Счётное (конечное/бесконечное)	Число заявок в очереди, состояние сети (ON/OFF)
Непрерывные состояния	Континуум (интервал, $\mathbb{R}^n$)	Температура, цена акции, координата частицы

Примеры: Процесс броуновского движения имеет непрерывные состояния (координата частицы), а число кликов на сайте — дискретные.

2. По типу временного множества T (табл. 9). В таблице 10 приведены варианты комбинаций типов состояний и временных множеств.

Таблица 9. Классификация моделей по типу временного множества

Тип	Множество T	Примеры
Дискретное время	$T = 0, 1, 2, ...$	Цена акции на конец дня, данные датчиков раз в час
Непрерывное время	$T = [0, +\infty)$ или $\mathbb{R}$	Изменение давления газа, сигнал в телекоммуникационном канале

Таблица 10. Комбинации типов

Время / Состояния	Дискретные состояния	Непрерывные состояния
Дискретное время	Цепи Маркова (ДЦМ), ARMA-модели	Авторегрессия (AR), Процессы с непрерывными состояниями
Непрерывное время	Цепи Маркова (НЦМ), Пуассоновский процесс	Винеровский процесс, Стохастические дифузии

Примеры моделей:

1. Дискретное время + дискретные состояния:

- Модель: Цепь Маркова с дискретным временем (ДЦМ).

- Приложение: Прогноз погоды (состояния: "дождь", "ясно", "облачно") ежедневно.

2. Непрерывное время + дискретные состояния:

- Модель: Пуассоновский процесс $N(t)$ — число событий к моменту t.

- Приложение: Число вызовов в call-центре за интервал $[0, t]$.

3. Дискретное время + непрерывные состояния:

- Модель: Авторегрессионный процесс 1-го порядка: $X_t = a \cdot X_{t-1} + \varepsilon_t$.

- Приложение: Прогноз курса валюты по дням.

4. Непрерывное время + непрерывные состояния:

- Модель: Винеровский процесс (броуновское движение) $W(t)$.

- Приложение: Моделирование траектории молекулы в жидкости.

Важные свойства для классификации

1. Стационарность:

- Строгая: Распределение $X(t_1), \ldots, X(t_k)$ не зависит от сдвига времени.

- Слабая: Моменты ($E[X(t)], Cov(X(t), X(s))$) инвариантны к сдвигу.

2. Марковское свойство:

Будущее зависит только от настоящего, а не от прошлого:

$$P(X(t_n) = x_n \mid X(t_1) = x_1, \ldots, X(t_{n-1}) = x_{n-1}) = P(X(t_n) = x_n \mid X(t_{n-1}) = x_{n-1}).$$

3. Эргодичность:

Усреднение по времени равно усреднению по ансамблю реализаций.

Значение для моделирования систем

1. Адекватность описания:

- Дискретные состояния подходят для систем с чёткими режимами (работа/сбой).

- Непрерывное время необходимо для моделирования "реального времени" (физические процессы).

2. Выбор метода анализа:

- Дискретное время → Рекуррентные уравнения.

- Непрерывное время → Дифференциальные уравнения (Колмогорова).

3. Интерпретируемость:

Марковские процессы (дискретные состояния) проще анализировать, чем общие СП.

2.1.3. Марковские процессы: свойство Маркова и основные типы цепей

Марковские процессы — фундаментальный класс случайных процессов, где будущее состояние системы зависит только от её текущего состояния, а не от истории. Это свойство (отсутствие последействия) делает их мощным инструментом для моделирования динамических систем со случайными переходами.

Свойство Маркова: формальное определение

Свойство Маркова (или марковское свойство) для стохастического процесса формально выражается следующим образом:

Стохастический процесс $X = X_t\big|_{t \in T}$ обладает свойством Маркова, если для любого момента времени t, будущие значения процесса (все X_s для $s>t$) при условии настоящего состояния Xt не зависят от прошлых состояний X_u для $u<t$.

Для процесса $X(t)$ с множеством состояний S свойство Маркова выполняется, если:

$$P\left(X(t_{n+1})=x_{n+1}\middle|X(t_1)=x_1,\dots,X(t_n)=x_n\right)=P\left(X(t_{n+1})=x_{n+1}\middle|X(t_n)=x_n\right) \qquad (104)$$

Условие: $t_1<t_2<\dots<t_n<t_{n+1}$ (упорядоченные моменты времени).

Смысл: Вся информация о будущем заключена в текущем состоянии. История не влияет на вероятности переходов.

Пример: Перемещение робота по карте. Вероятность перехода в соседнюю ячейку зависит только от текущего положения, а не от пройденного пути.

Основные типы марковских процессов

Классификация основана на природе времени и состояний:

1. Цепи Маркова с дискретным временем (ДЦМ)

- Время: $T=0,1,2,\dots$ (шаги).

- Состояния: Дискретные (S — конечное или счётное множество: $s_1, s_2, \dots$).

- Динамика: Переходы происходят в фиксированные моменты.

- Матрица переходных вероятностей:

$P=(p_{ij})$, где $p_{ij}=P\left(X_{n+1}=j\middle|X_n=i\right)$

Свойства:

- $0 \le p_{ij} \le 1$

- $\sum_{j \in S} p_{ij} = 1$ (сумма по строке).

- Граф состояний:

Узлы — состояния, рёбра $i \Rightarrow j$ помечены вероятностями $p_{ij} > 0$.

- Пример: Игральная кость. Состояние — текущая сумма очков. Переходы — бросок кости.

2. Цепи Маркова с непрерывным временем (НЦМ)

- Время: $T = [0, +\infty)$.

- Состояния: Дискретные (аналогично ДЦМ).

- Динамика: Переходы происходят в случайные моменты времени.

- Интенсивности переходов:

$q_{ij} \ge 0$ — скорость перехода из i в j ($i \ne j$).

- Матрица интенсивностей (Q-матрица):

$Q = (q_{ij})$, где:

- q_{ij} — интенсивность перехода из i в j ($i \ne j$),

- $q_{ii} = -\sum_{j \ne i} q_{ij}$ (сумма по строке равна 0).

- Время пребывания в состоянии i:

Экспоненциально распределено с параметром $\lambda_i = -q_{ii}$.

- Пример: Работа технического устройства. Состояния: {0={сбой}, 1={работа}}. Интенсивности: q_{01} (восстановление), q_{10} (отказ).

Сравнение ДЦМ и НЦМ приведено в таблице 11

Таблица 11. Сравнение дискретных и непрерывных цепей Маркова

Характеристика	ДЦМ	НЦМ
Временная шкала	Дискретные шаги (n=0,1,2,dots)	Непрерывное время (t geq 0)
Переходы	В фиксированные моменты	В случайные моменты
Описание динамики	Матрица P (вероятности)	Матрица Q (интенсивности)
Время между переходами	Детерминировано (1 шаг)	Экспоненциальное распределение
Уравнения эволюции	Рекуррентные уравнения	Дифференциальные уравнения Колмогорова

Ключевые приложения

1. ДЦМ:

- Моделирование дискретных стратегий (обучение с подкреплением).

- Прогнозирование на дискретных временных рядах (финансы, биология).

2. НЦМ:

- Системы массового обслуживания (динамика очередей).

- Надёжность технических систем (переходы между состояниями работоспособности).

- Биохимические реакции (моделирование случайных взаимодействий).

Значение для моделирования

- Вычислительная эффективность: Свойство Маркова сокращает объём данных для прогнозирования (не нужна полная история).

- Аналитическая доступность: Для марковских цепей существуют методы расчёта:

- Стационарных распределений (предельные вероятности).

- Времени достижения состояний.

- Поглощающих вероятностей.

- База для сложных моделей: НЦМ лежат в основе процессов рождения-гибели, сетей Петри, стохастических дифференциальных уравнений.

2.1.4. Потоки событий: определение и пуассоновский поток

Поток событий — последовательность однородных событий, наступающих в случайные моменты времени:

$$\{t_1, t_2, t_3, \ldots\}, \text{ где } 0 \le t_1 < t_2 < t_3 < \ldots$$

Примеры: вызовы в call-центре, отказы оборудования, транзакции в банке, прибытие самолётов.

Простейший (пуассоновский) поток

Пуассоновский поток — базовая математическая модель для событий, удовлетворяющая трём критериям:

1. Стационарность

- Вероятность наступления k событий на интервале $(\tau, \tau + t)$ зависит только от длины t, но не от положения τ на временной оси:

$$P(k, t) = f(k, t)$$

- Следствие: Среднее число событий за время t равно λt, где λ — интенсивность потока.

2. Ординарность (отсутствие кратных событий)

- Вероятность наступления двух и более событий за малый интервал Δt пренебрежимо мала:

$$P(\ge 2, \Delta t) = o(\Delta t) \text{ при } \Delta t \to 0.$$

- Практический смысл: События приходят поодиночке, а не группами.

3. Отсутствие последействия

- Число событий на непересекающихся интервалах времени независимо:

$$P(A \cap B) = P(A)P(B), \text{ где } A \text{ — событие на } (t_1, t_2), B \text{ — на } (t_3, t_4).$$

- Интерпретация: Прошлые события не влияют на будущие.

Интенсивность потока λ

- Определение: Среднее число событий в единицу времени: $\lambda = E[N(1)]$.

- Расчёт:

  - Для интервала длиной t: $E[N(t)] = \lambda t$.

  - При $\Delta t \to 0$: $P(\textit{ровно 1 событие за } \Delta t) \approx \lambda \Delta t$.

- Физический смысл: Параметр, характеризующий "плотность" событий (например, 5 вызовов/час).

Ключевые свойства пуассоновского потока

1. Распределение числа событий:

Число событий $N(t)$ за время t имеет распределение Пуассона:

$$P(N(t)=k)=\frac{(\lambda t)^k e^{-\lambda t}}{k!}, k=0,1,2,\ldots \quad (105)$$

Пример: При λ = 3 события/час, вероятность ровно 2 события за 1 час: $P(N(1)=2)=(3^2 e^{-3})/2! \approx 0.224.$

2. Распределение интервалов между событиями:

Время τ между соседними событиями имеет экспоненциальное распределение:

$$F_\tau(t)=1-e^{-\lambda t},$$
$$f_\tau(t)=\lambda e^{-\lambda t} \quad (106)$$
$$(t\geq 0)$$

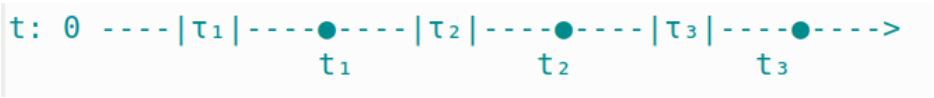
Следствие: $E[tau]=1/\lambda$ (средний интервал).

3. Связь с экспоненциальным распределением:

- Если интервалы между событиями $\tau_1, \tau_2, \ldots$ — независимые экспоненциальные СВ с параметром λ, то поток — пуассоновский.

Визуализация свойств

Временная шкала пуассоновского потока:



где:

● — событие

|τᵢ| — интервал времени $\sim \exp(\lambda)$

Значение для моделирования систем

1. Моделирование входных потоков:

- В системах массового обслуживания (СМО) пуассоновский поток заявок (M) — стандартное допущение .

2. Аналитическая простота:

- Свойство отсутствия последействия позволяет применять марковские модели.

3. Предельные теоремы:

Суммарный поток от большого числа независимых редких источников аппроксимируется пуассоновским (теорема Пальма).

Реальные потоки могут нарушать свойства:

- Телефонные вызовы утром → нестационарны.

- Пакетная передача данных → неординарна.

Таким образом, пуассоновский поток — идеализированная, но мощная модель для анализа систем с хаотичными событиями. Его свойства позволяют строить аналитические решения для СМО, а экспоненциальное распределение интервалов обеспечивает марковость процесса.

2.2. Дискретные цепи Маркова (ДЦМ)

2.2.1. Определение, матрица переходных вероятностей, граф состояний.

Формальное определение ДЦМ

Дискретная цепь Маркова (ДЦМ) — это случайный процесс с дискретным временем $X_n, n=0,1,2,\ldots$, принимающий значения в дискретном множестве состояний $S=\{s_1, s_2, \ldots\}$, удовлетворяющий марковскому свойству:

$$P\left(X_{n+1}=j \middle| X_0=i_0, X_1=i_1, \ldots, X_n=i\right)=P\left(X_{n+1}=j \middle| X_n=i\right) \quad (107)$$

где:

- $i, j \in S$ — текущее и следующее состояния,

- $i_0, i_1, \ldots$ — история процесса.

То есть, вероятность перехода в будущее состояние зависит только от текущего состояния и не зависит от предшествующих состояний процесса.

Ключевое допущение: Цепь однородна, если переходные вероятности не зависят от времени n.

Матрица переходных вероятностей

Основной инструмент описания динамики ДЦМ — стохастическая матрица размером $|S|\times|S|$:

$$P=(p_{ij}), \quad (108)$$

где $p_{ij}=P\left(X_{n+1}=j \middle| X_n=i\right)$

Свойства матрицы P:

1. $0 \le p_{ij} < 1$ (вероятности),

2. $\sum_{j \in S} p_{ij} = 1$ (сумма элементов каждой строки равна 1).

Пример матрицы для $S = \{A, B\}$:

$$P = \begin{pmatrix} p_{AA} & p_{AB} \\ p_{BA} & p_{BB} \end{pmatrix} = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix}$$

Интерпретация:

- Из состояния A вероятность остаться в A — 70%, перейти в B — 30%.

- Из состояния B вероятность перейти в A — 40%, остаться в B — 60%.

Граф состояний

Визуальное представление ДЦМ в виде ориентированного взвешенного графа:

- Узлы: Состояния $s_i \in S$.

- Рёбра: Переход $i \to j$ существует, если $p_{ij} > 0$.

- Веса рёбер: Вероятности перехода p_{ij}.

Правила построения (табл. 12):

1. Петля $i \to i$ рисуется при $p_{ii} > 0$.

2. Пропуск ребра означает $p_{ij} = 0$ (переход невозможен).

3. Сумма весов выходящих рёбер из узла равна 1.

Таблица 12. Связь между матрицей и графом

Элемент матрицы	Соответствие в графе
p_{ij}	Вес ребра $i \to j$
Строка i	Исходящие рёбра из узла i
$p_{ii} = 0$	Отсутствие петли у узла i
Нулевой столбец j	Узел j — сток (нет исходящих рёбер)

Пример: блуждание по числовой прямой

Система: Частица в точках $S = \{-1, 0, 1\}$.

Правила:

- Из 0: С равной вероятностью переходит в -1 или 1.

- Из -1 и 1: Остаётся на месте с вероятностью 1.

Матрица переходов:

$$P=\begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 0 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}$$

Значение для анализа систем

1. Матрица P позволяет:

- Рассчитать вероятности за k шагов: $P^{(k)}=P^k$.

- Найти стационарное распределение (решение $\pi=\pi P$, при котором вектор вероятностей π не изменяется во времени при применении матрицы P).

2. Граф состояний помогает визуально выявить:

- Коммуникативные классы (связные компоненты).

- Поглощающие состояния (узлы без исходящих рёбер).

- Циклы и эргодичность.

2.2.2. Классификация состояний (достижимость, возвратность, периодичность, поглощающие).

Классификация состояний ДЦМ позволяет понять глобальную структуру цепи и предсказать её долгосрочное поведение. Основные категории иллюстрируются через граф состояний и анализируются с помощью матрицы переходов (P). В рамках анализа ДЦМ состояния разделяют по ряду фундаментальных признаков, которые позволяют понимать долгосрочное поведение цепи и структуру переходов.

1. Достижимость (Accessibility)

Говорят, что состояние j достижимо из состояния i, если существует конечная вероятность попасть из i в j за некоторое число шагов, то есть $P_{ij}^{(n)}>0$ для некоторого $n\geq 1$, где $P_{ij}^{(n)}$ — вероятность оказаться в состоянии j через n шагов, начав из i.

Состояния разбиваются на коммуницирующие классы: i и j коммуницируют, если каждый достижим из другого.

2. Возвратность и невозвратность

- Возвратное состояние (recurrent): вероятность вернуться в состояние i, начав из i, равна 1.

- Невозвратное (transient): вероятность вернуться в состояние менее 1.

Возвратные состояния характеризуются “бесконечным посещением” в длинном процессе, невозвратные — возможностью покинуть их навсегда.

3. Периодичность состояния

Период состояния $d(i)$ — наибольший общий делитель всех n, для которых $P_{ii}^{(n)}>0$ (т.е. можно вернуться в i за n шагов).

- Если $d(i)=1$, состояние апериодично (возврат возможен в любые большие промежутки времени).

- Если $d(i)>1$, состояние периодично (возврат возможен только через число шагов, кратное $d(i)$).

4. Поглощающие состояния

Поглощающее (absorbing) состояние i, если $P_{ii}=1$ (попав в него, цепь не покидает этого состояния).

Примеры: в моделях надежности — “неисправимо испорченное” оборудование; в очередях — “система закрыта”.

Классификация выше используется для анализа долгосрочного поведения системы, поиска стационарного распределения и изучения свойств устойчивости и предсказуемости для реальных приложений цепей Маркова.

2.2.3. Предельные вероятности. Стационарное распределение: условия существования и единственности. Уравнения для стационарных вероятностей.

Предельные вероятности (или стационарные вероятности) — это значения вероятностей нахождения марковской цепи в различных состояниях, к которым стремятся вероятности при устремлении времени к бесконечности, независимо от начального состояния. То есть, если существует предел

$$\pi_j=\lim_{n\to\infty} P\left(X_n=j\middle|X_0=i\right)$$

и этот предел не зависит от начального состояния i, то π_j называется предельной (стационарной) вероятностью состояния j.

Стационарное распределение — это такой вектор вероятностей $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, который не изменяется во времени при переходах по матрице вероятностей P:

$$\pi = \pi P$$

Или, по компонентам:

$$\pi_j = \sum_i \pi_i p_{ij}$$

для всех j, где p_{ij} — вероятность перехода из состояния i в состояние j.

Условия:

- $\pi_j \geq 0$ для всех j;
- $\sum_j \pi_j = 1.$

Стационарное распределение существует и единственно тогда и только тогда, когда выполняются следующие условия:

- Цепь неприводима (любой класс состояний достижим из любого другого — все состояния сообщаются).

- Цепь апериодична (нет жесткой цикличности возвратов в состояния).

- Положительная возвратность (ожидаемое время возврата в состояние конечно).

Тогда для конечной однородной ДЦМ вероятности состояний с ростом времени сходятся к стационарному распределению, независимо от начальных условий.

Для поиска стационарного распределения необходимо решить систему:

$$\begin{cases} \pi_j = \sum_{i=1}^{n} \pi_i p_{ij}, \quad j = 1, \ldots, n \\ \sum_{j=1}^{n} \pi_j = 1 \end{cases} \tag{109}$$

Обычно одна из строк системы убирается, а вместо нее добавляется нормировочное условие, чтобы система была невырожденной.

Таким образом, предельные вероятности ДЦМ — это значения вероятностей пребывания в состояниях системы при длительном наблюдении, определяемые стационарным распределением, которое можно найти как решение системы линейных уравнений $\pi = \pi P$ с нормировкой. Существование и единственность гарантируют неприводимость и апериодичность цепи.

2.2.4. Применение ДЦМ в моделировании: модели надежности, маркетинговые модели (переключение брендов), простые модели очередей

Дискретные цепи Маркова (ДЦМ) — мощный инструмент для анализа систем с дискретными состояниями и стохастическими переходами. Рассмотрим ключевые прикладные области.

1. Модели надежности технических систем

Задача: Оценка вероятности безотказной работы системы с резервированием и ремонтом.

Пример: система с "горячим" резервом

Состояния:

S = {0: Оба элемента работают, 1: Один работает, один в ремонте, 2: Отказ (оба нерабочие)}

Параметры:

λ — интенсивность отказа элемента,

μ — интенсивность ремонта.

Матрица переходов за шаг Δt:

$$P=\begin{pmatrix} 1-2\lambda\Delta t & 2\lambda\Delta t & 0 \\ \mu\Delta t & 1-(\lambda+\mu)\Delta t & \lambda\Delta t \\ 0 & 0 & 1 \end{pmatrix}$$

Расчет характеристик:

- Вероятность отказа: π_2 (стационарный режим).

- Коэффициент готовности: $1-\pi_2$.

Анализ: Решение $\pi P=\pi$ позволяет оптимизировать число резервных элементов.

2. Маркетинговые модели переключения брендов

Задача: Прогноз доли рынка при миграции потребителей между брендами.

Пример: конкуренция трех брендов

Состояния:

S = {A, B, C} (лояльность бренду).

Данные: Ежемесячные переходы (из опросов):

- 80% клиентов A остаются, 10% переходят к B, 10% к C.

- 70% клиентов B остаются, 20% → A, 10% → C.

- 90% клиентов C остаются, 5% → A, 5% → B.

-Матрица переходов:

$$P=\begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.2 & 0.7 & 0.1 \\ 0.05 & 0.05 & 0.9 \end{pmatrix}$$

Стационарное распределение: Решение $\pi = \pi P$ дает долгосрочные доли рынка:

$$\pi_A \approx 0.33, \quad \pi_B \approx 0.22, \quad \pi_C \approx 0.45$$

Управление: Влияние скидок (изменение p_{AB}, p_{AC}) на π_A.

3. Простые модели очередей

Задача: Анализ одноканальной очереди с дискретным временем.

Пример: буфер роутера

Состояния: S = {0, 1, 2, ..., N} (число пакетов в буфере).

Правила:

За шаг Δt:

- С вероятностью α прибывает новый пакет (если есть место).

- С вероятностью β обрабатывается один пакет (если очередь 0).

Матрица переходов (табл. 13):

Таблица 13. Матрица переходов к задаче об одноканальной очереди

Действие	Переход	Переход
Прибытие + обработка	i → i	αβ
Только прибытие	i → i+1	α(1-β)
Только обработка	i → i-1	(1-α)β
Ничего	i → i	(1-α)(1-β)

Поглощающее состояние: i = N (переполнение буфера).

Расчеты:

- Вероятность переполнения: π_N.

- Средняя длина очереди: $\sum_{i=0}^{N} i\,\pi_i$.

Типовые алгоритмы анализа

1.Расчет стационарного распределения:

```
import numpy as np
P = np.array([[0.8, 0.1, 0.1], [0.2, 0.7, 0.1], [0.05, 0.05, 0.9]])  # Маркетинговый пример
eigenvalues, eigenvectors = np.linalg.eig(P.T)   #Транспонированная матрица
pi = eigenvectors[:, np.isclose(eigenvalues, 1)]   #Собственный вектор для λ=1
pi = pi / pi.sum()   Нормировка
```

2.Прогноз на k шагов:

- $P^{(k)} = P^k$ (матрица в степени k).

- Вероятность перехода из i в j заk шагов: $\left(P^k\right)[i, j]$.

Заключение

ДЦМ предоставляют унифицированный фреймворк для:

- Оценки надежности систем с ремонтом и резервированием,

- Прогнозирования динамики рынка и оптимизации маркетинга,

- Моделирования простых очередей с дискретными событиями.

2.3. Непрерывные цепи Маркова (НЦМ) и процессы рождения-гибели

2.3.1. Определение, интенсивности переходов, матрица интенсивностей

Непрерывная цепь Маркова (НЦМ) — это стохастический процесс с дискретным (конечным или счетным) пространством состояний и непрерывным временем, обладающий марковским свойством: вероятность перейти в следующее состояние зависит только от текущего состояния и не зависит от всей предыстории процесса. Переходы между состояниями могут происходить в любые (случайные) моменты времени, а не только в дискретные.

Формально: если $X(t)$ — состояние процесса в момент времени t, то для любых $s>0$:

$$P\left(X(t+s)=j \,\middle|\, X(u), u \le t\right) = P\left(X(t+s)=j \,\middle|\, X(t)\right) \tag{110}$$

Интенсивность перехода (или параметр перехода, rate) из состояния i в состояние j ($i \neq j$) — это мгновенная скорость перехода, характеризующая "частоту", с которой из i происходит переход в j:

$$q_{ij} = \lim_{h \to 0} \frac{P\left(X(t+h)=j \,\middle|\, X(t)=i\right)}{h} \tag{111}$$

Интенсивности $q_{ij} \geq 0$: их сумма по всем возможным j (для фиксированного i) определяет суммарную вероятность покинуть состояние i в единицу времени.

Для диагонали:

$$q_{ii} = -\sum_{j \neq i} q_{ij} \tag{112}$$

Таким образом, сумма по строке матрицы интенсивностей всегда равна нулю: $\sum_j q_{ij} = 0.$

Моменты времени пребывания в состоянии i подчинены экспоненциальному закону с параметром $-q_{ii}$.

Матрица интенсивностей (или матрица генератора $\boldsymbol{Q} = (q_{ij})$) — это квадратная матрица размерности числа состояний, где:

- Вне диагонали: q_{ij} — интенсивность перехода из i в j ($i \neq j$), все элементы неотрицательны.

- По диагонали: $q_{ii} = -\sum_{j \neq i} q_{ij}$, всегда отрицательны или нули.

- Сумма по каждой строке равна нулю: $\sum_j q_{ij} = 0.$

Эта матрица определяет динамику НЦМ через систему дифференциальных уравнений Колмогорова:

$$\frac{d}{dt}\boldsymbol{P}(t) = \boldsymbol{P}(t)\boldsymbol{Q} \tag{113}$$

Процессы рождения-гибели

Процесс рождения-гибели — частный пример НЦМ, в котором из любого состояния разрешены только переходы в "соседние": $k \to k+1$ (рождение) с интенсивностью λ_k , $k \to k-1$ (гибель) с интенсивностью μ_k ; остальные переходы невозможны ($q_{ij} = 0$ для всех остальных пар).

Матрица интенсивностей процессов рождения-гибели имеет тридиагональный вид:

$$\begin{pmatrix} -(\lambda_0) & \lambda_0 & 0 & 0 & \ldots \\ \mu_1 & -(\lambda_1+\mu_1) & \lambda_1 & 0 & \ldots \\ 0 & \mu_2 & -(\lambda_2+\mu_2) & \lambda_2 & \ldots \\ \ldots & \ldots & \ldots & \ldots & .. \end{pmatrix} \tag{114}$$

Такие процессы широко применяются в моделировании систем массового обслуживания, биологии, демографии и т.д. Их простота и аналитическая решаемость делают их базовой моделью для изучения систем с возможностью появления и исчезновения объектов или заявок.

Вывод:

Непрерывные цепи Маркова описываются матрицей интенсивностей, а сама динамика — с помощью экспоненциальных времен пребывания и вероятностей перехода между состояниями. Процессы рождения-гибели — важный и наглядный класс таких моделей.

2.3.2. Уравнения Колмогорова (прямые и обратные) для вероятностей состояний

В непрерывных цепях Маркова (НЦМ) динамика вероятностей пребывания системы в различных состояниях во времени описывается системой дифференциальных уравнений, называемых уравнениями Колмогорова.

Пусть $P_{ij}(t)=P\left(X(t)=j \middle| X(0)=i\right)$ — вероятность того, что в момент времени t система находится в состоянии j, если в начале ($t=0$) была в состоянии i. Прямые уравнения Колмогорова (уравнения вперед) уравнения описывают производные вероятностей по времени, выражая эволюцию вероятностей состояний через переходные интенсивности (матрицу генератора Q):

$$\frac{d}{dt}P_{ij}(t)=\sum_k P_{ik}(t)q_{kj} \tag{115}$$

где:

- q_{kj} — интенсивность перехода из состояния k в состояние j ($q_{kk}=-\sum_{j\neq k} q_{kj}$),
- $P_{ij}(0)=\delta_{ij}$ (кронекерова дельта: в начале с вероятностью 1 система в состоянии i).

В векторно-матричном виде для всей матрицы переходных вероятностей:

$$\frac{d}{dt}\boldsymbol{P}(t)=\boldsymbol{P}(t)\boldsymbol{Q} \tag{116}$$

Обратные уравнения Колмогорова (или уравнения назад) описывают производную по времени вероятности, но по начальному состоянию:

$$\frac{d}{dt}P_{ij}(t)=\sum_k q_{ik}P_{kj}(t) \tag{117}$$

Аналогично, в матричном виде:

$$\frac{d}{dt}\boldsymbol{P}(t)=\boldsymbol{Q}\,\boldsymbol{P}(t) \tag{118}$$

В процессах рождения-гибели уравнения Колмогорова конкретизируются для случая, когда разрешены только переходы $k\to k+1$ (рождение) с интенсивностью λ_k и $k\to k-1$ (гибель) с интенсивностью μ_k:

Прямые (вперед) уравнения для вероятности $p_n(t)=P\left(X(t)=n\middle|X(0)=i\right)$:

$$\frac{dp_n(t)}{dt}=\lambda_{n-1}\,p_{n-1}(t)+\mu_{n+1}\,p_{n+1}(t)-(\lambda_n+\mu_n)\,p_n(t) \tag{119}$$

Начальные условия: $p_n(0)=1$ если $n=i$, иначе 0.

Заключение

Прямые уравнения выражают темпы изменения вероятностей через текущие вероятности всех состояний и параметры генератора Q.

Обратные уравнения выражают это через вероятности перехода от разных начальных состояний.

Они — фундамент для расчёта временных и стационарных характеристик НЦМ и конкретных моделей, таких как процессы рождения-гибели.

Эти уравнения лежат в основе численного и аналитического анализа модельных систем в инженерии, биологии, массовом обслуживании, вычислительных системах и других прикладных областях.

2.3.3. Стационарный режим: уравнения для стационарных вероятностей

В непрерывных цепях Маркова (НЦМ) стационарный режим означает, что вероятности пребывания системы в различных состояниях не изменяются со временем. Это соответствует установившемуся поведению системы, при котором притоки и оттоки вероятностей для каждого состояния уравновешены.

Пусть p_j — стационарная вероятность нахождения системы в состоянии j при $t\to\infty$. Тогда в стационарном режиме вероятности не зависят от времени, и выполняются следующие уравнения:

$$\frac{dp_j(t)}{dt}=0,\quad \forall\, j \tag{120}$$

где $p_j(t)$ — вероятность нахождения в состоянии j в момент времени t.

Для матрицы интенсивностей $Q=(q_{ij})$ (матрица генератора), уравнения стационарного режима для всех j записываются так:

$$\sum_i p_i, q_{ij}=0, \quad \forall j \tag{121}$$

То есть, сумма притока вероятности в состояние j (по всем i) минус отток из j равна нулю. Это условие баланса вероятности.

Добавляется также условие нормировки вероятностей: $\sum_j p_j=1$ В матричном виде это записывается как: $\boldsymbol{p}\cdot Q=0$, где $\boldsymbol{p}$ — строка-вероятностей (стационарное распределение), Q — матрица интенсивностей (генератор).

Для процессов рождения-гибели, в которых разрешены только переходы между соседними состояниями (например, от k к $k+1$ с интенсивностью λ_k и от k к $k-1$ с интенсивностью μ_k), стационарные уравнения принимают рекуррентный вид:

$$p_k \lambda_k = p_{k+1} \mu_{k+1}, \quad \forall k \geq 0 \tag{122}$$

или полный баланс:

$$p_k(\lambda_k+\mu_k)=p_{k-1}\lambda_{k-1}+p_{k+1}\mu_{k+1} \tag{123}$$

с учетом граничных условий (например, для $k=0$ у некоторых моделей $\mu_0=0$).

Суммарное условие:

$$\sum_{k=0}^{\infty} p_k=1 \tag{124}$$

Заключение

- Уравнения для стационарных вероятностей в НЦМ выражают равенство притоков и оттоков вероятности для каждого состояния.

- Для любой непрерывной цепи Маркова стационарный режим описывается системой линейных уравнений: $\boldsymbol{p}Q=0$ вместе с $\sum p_j=1$.

- В процессах рождения-гибели эти уравнения просты для построения рекурсии и аналитического/численного решения.

Стационарное распределение позволяет находить долю времени, которую система в среднем проводит в каждом состоянии, и рассчитывать устоявшиеся эксплуатационные и сервисные характеристики моделируемых систем.

2.3.4. Процессы рождения-гибели

1. Определение процесса рождения-гибели

Процесс рождения-гибели — один из важнейших видов непрерывных цепей Маркова с дискретными состояниями {0, 1, 2, ...}. Такой процесс описывает системы, где за короткие интервалы времени возможны только две элементарные перемены:

- "Рождение" — переход из состояния n в $n+1$ (например, появление новой заявки в очереди);

- "Гибель" — переход из состояния n в $n-1$ (например, обслуживание или уход заявки).

Процесс полностью определяется интенсивностями (параметрами) рождений λ_n и гибелей μ_n, зависящими от текущего числа объектов (состояния) n.

2. Граф состояний

Состояния изображаются вершинами графа (0, 1, 2, ...), переходы между ними — направленными стрелками (рис. 18):

- Из состояния n вправо ($n \to n+1$) со скоростью λ_n — рождение.

- Влево (n → n-1) со скоростью μ_n — гибель.

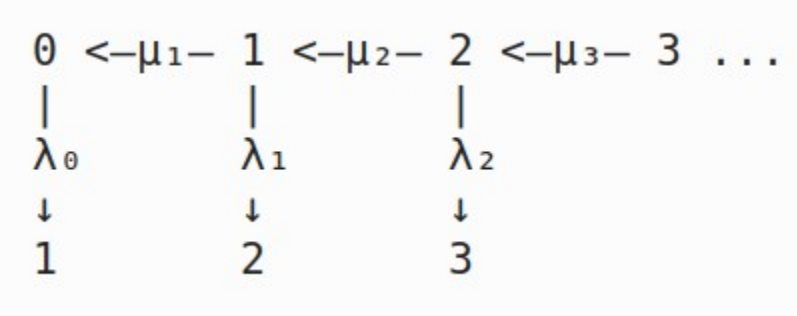


Рисунок 18 - Граф состояний

Границы могут быть отражающими или поглощающими в зависимости от модели.

3. Условия существования стационарного режима

Стационарное распределение — это такие вероятности p_n пребывания в каждом n-ом состоянии, которые остаются неизменными во времени. Для существования стационарного режима при счетном числе состояний необходима сходимость ряда:

$$\sum_{k=1}^{\infty} \prod_{i=1}^{k} \frac{\lambda_{i-1}}{\mu_i} < \infty \qquad (125)$$

Если это условие выполняется, процесс называется эргодическим, и существует единственное стационарное распределение.

4. Формулы для стационарных вероятностей

Стационарные вероятности p_n в процессе рождения-гибели выражаются через рекуррентные соотношения баланса потоков:

$$\lambda_{n-1} p_{n-1} = \mu_n p_n \tag{126}$$

или

$$p_n = \frac{\lambda_{n-1}}{\mu_n} p_{n-1} \tag{127}$$

Решение для всех n:

$$p_n = p_0 \prod_{i=1}^{n} \frac{\lambda_{i-1}}{\mu_i}, \quad n \geq 1 \tag{128}$$

где p_0 определяется из условия нормировки:

$$p_0 = \left(1 + \sum_{k=1}^{\infty} \prod_{i=1}^{k} \frac{\lambda_{i-1}}{\mu_i} \right)^{-1} \tag{129}$$

Таким образом, стационарные вероятности для каждого состояния можно выписать явно при заданных интенсивностях рождения и гибели, если выполняется условие нормировки.

2.4. Системы массового обслуживания (СМО) - как основной класс стохастических моделей

Системы массового обслуживания (СМО) представляют фундаментальный класс стохастических моделей, предназначенных для анализа процессов, где заявки (требования) поступают в случайные моменты времени и обслуживаются специальными устройствами (каналами). Эти модели находят применение в телекоммуникациях, транспорте, производстве, компьютерных сетях и других областях, где возникает конкуренция за ограниченные ресурсы.

2.4.1. Основные элементы СМО

Ключевые элементы СМО:

1. Входящий поток: Последовательность заявок, поступающих в систему. Характеризуется:

- Интенсивностью поступления заявок (λ)

- Законом распределения интервалов между заявками

- Свойствами (ординарность, стационарность, отсутствие последействия)

2. Очередь: Место ожидания заявок, когда все каналы заняты. Характеризуется:

- Дисциплиной обслуживания (FIFO, LIFO, приоритетная)

- Ограниченной или неограниченной длиной

3. Каналы обслуживания: Устройства, выполняющие обслуживание заявок. Характеризуются:

- Количеством каналов (m)

- Интенсивностью обслуживания (μ)

- Законом распределения времени обслуживания

4. Выходящий поток: Поток обслуженных заявок, покидающих систему

Для описания структуры СМО используется универсальная нотация - классификация Кендалла в виде A/B/m/n/K :

- A: Распределение интервалов между заявками

(M - экспоненциальное, D - детерминированное, E_k - Эрланга, G - произвольное)

- B: Распределение времени обслуживания

(обозначения аналогичны A)

- m: Количество каналов обслуживания (целое число ≥ 1)

- n: Максимальное число заявок в системе (включая обслуживаемые)

- K: Размер популяции (источника заявок)

Примеры обозначений:

- M/M/1: Экспоненциальные входящий поток и обслуживание, один канал, бесконечная очередь

- M/D/2/10: Пуассоновский входящий поток, детерминированное обслуживание, 2 канала, максимум 10 заявок в системе

2.4.2. Характеристики эффективности СМО

1. Вероятностные характеристики:

- $P_{отк}$ - вероятность отказа (потери заявки)

- $P_{оч}$ - вероятность ожидания в очереди

- P_k - вероятность нахождения в системе k заявок

2. Показатели загрузки системы:

- $\rho = \lambda / (m\mu)$ - коэффициент загрузки системы

- $\rho_{канал} = \lambda / (m\mu)$ - средняя загрузка одного канала

3. Средние числовые характеристики:

- L_s - среднее число заявок в системе

- L_q - среднее число заявок в очереди

- W_s - среднее время пребывания заявки в системе

- W_q - среднее время ожидания в очереди

4. Пропускная способность:

- $A = \lambda (1 - P_{отк})$ - абсолютная пропускная способность

2.4.3. Аналитические модели

Одноканальные СМО:

1. M/M/1/∞ (с неограниченной очередью):

- Условие стационарности: $\rho < 1$

- Основные характеристики:

  - $P_0 = 1 - \rho$

  - $P_n = (1 - \rho)\rho^n$

  - $L_s = \rho / (1 - \rho)$

  - $W_s = 1 / [\mu (1 - \rho)]$

2. M/M/1/0 (с отказами):

- $P_0 = 1 / (1 + \rho)$

- $P_{отк} = \rho / (1 + \rho)$

- $A = \lambda(1 - P_{отк}) = \lambda/(1+\rho)$

Многоканальные СМО:

1. M/M/m/∞:

- Условие стационарности: $\rho < 1$
- Вероятность простоя каналов:

$$P_0 = [\Sigma_{k=0}^{m-1} (m\rho)^k / k! + (m\rho)^m / (m!(1-\rho))]^{-1}$$

- Вероятность ожидания: $C(m,\rho) = [(m\rho)^m / (m!(1-\rho))] P_0$
- $L_q = [\rho C(m,\rho)]/(1-\rho)$

2. M/M/m/0 (система с отказами):

- Формула Эрланга для вероятности отказа:

$$P_{отк} = [(m\rho)^m / m!] / [\Sigma_{k=0}^{m} (m\rho)^k / k!]$$

СМО с ограниченной очередью M/M/m/n:

- Сочетает черты систем с очередью и отказами
- При n > m имеем очередь ограниченной длины
- При заполнении всех мест в очереди (n заявок в системе) новые заявки теряются

СМО с ненадежными каналами:

- Каналы могут выходить из строя с определенной интенсивностью
- Время восстановления канала имеет экспоненциальное распределение
- Требует расширенного анализа с учетом состояний каналов

2.4.4. Основные допущения и границы применимости аналитических моделей

Аналитические модели СМО базируются на следующих допущениях:

1. Пуассоновский входящий поток (экспоненциальное распределение интервалов)
2. Экспоненциальное распределение времени обслуживания
3. Стационарность параметров системы (λ и μ постоянны)

4. Эргодичность системы (существование стационарного режима)

5. Независимость времени обслуживания от входящего потока

Границы применимости:

1. Для непуассоновских потоков (G/M/m, M/G/m) решения существуют только для отдельных случаев

2. При наличии приоритетов в обслуживании требуются специальные модели

3. Системы с коррелированными входящими потоками или временами обслуживания

4. СМО с неоднородными заявками или каналами

5. Сетевые системы массового обслуживания

6. Системы с изменяющимися параметрами (нестационарные СМО)

В случаях нарушения основных допущений аналитические модели теряют точность, и для анализа эффективности СМО необходимо применять методы имитационного моделирования, позволяющие учесть сложные зависимости и неэкспоненциальные распределения.

2.4.5. Пример системы: СМО M/M/1/0 (одноканальная система с отказами)

Описание системы:

- Клиенты прибывают в банковское отделение с единственным оператором

- Если оператор свободен - обслуживание начинается немедленно

- Если оператор занят - клиент уходит без обслуживания

- Время между прибытиями: экспоненциальное (λ = 4 клиента/час)

- Время обслуживания: экспоненциальное (μ = 5 клиентов/час)

Программа на Python для моделирования СМО M/M/1/0

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import expon, poisson

def simulate_mm1_queue(lambda_rate, mu_rate, simulation_time):
    """
    Параметры:
    lambda_rate - интенсивность входящего потока (заявок/час)
    mu_rate     - интенсивность обслуживания (заявок/час)
    simulation_time - время моделирования (часы)
    """
    # Статистика
```

```
total_arrivals = 0
total_served = 0
total_rejected = 0
server_busy = False

# Время следующего события
next_arrival = expon.rvs(scale=1/lambda_rate)
next_departure = float('inf')

current_time = 0

while current_time < simulation_time:
    # Обработка прибытия
    if next_arrival < next_departure:
        current_time = next_arrival
        total_arrivals += 1

        if not server_busy:
            server_busy = True
            total_served += 1
            next_departure = current_time + expon.rvs(scale=1/mu_rate)
        else:
            total_rejected += 1

        next_arrival = current_time + expon.rvs(scale=1/lambda_rate)

    # Обработка ухода
    else:
        current_time = next_departure
        server_busy = False
        next_departure = float('inf')

# Расчет характеристик
P_reject = total_rejected / total_arrivals
A = total_served / simulation_time  # Абсолютная пропускная способность

# Аналитические расчеты
rho = lambda_rate / mu_rate
P_reject_analytical = rho / (1 + rho)
A_analytical = lambda_rate * (1 - P_reject_analytical)

return {
    "simulated": {
        "total_arrivals": total_arrivals,
        "total_served": total_served,
        "total_rejected": total_rejected,
        "P_reject": P_reject,
        "A": A
    },
```

```
        "analytical": {
            "P_reject": P_reject_analytical,
            "A": A_analytical,
            "rho": rho
        }
    }

# Параметры моделирования
lambda_rate = 4    # 4 клиента в час
mu_rate = 5        # 5 клиентов в час
simulation_time = 1000  # 1000 часов моделирования

# Запуск симуляции
results = simulate_mm1_queue(lambda_rate, mu_rate, simulation_time)

# Вывод результатов
print("="*55)
print(f"Результаты моделирования СМО M/M/1/0 (t={simulation_time} ч.)")
print(f"Интенсивность входящего потока (λ): {lambda_rate} заявок/час")
print(f"Интенсивность обслуживания (μ): {mu_rate} заявок/час")
print("="*55)
print(f"Всего заявок: {results['simulated']['total_arrivals']}")
print(f"Обслужено: {results['simulated']['total_served']}")
print(f"Отклонено: {results['simulated']['total_rejected']}")
print(f"Вероятность отказа (модель): {results['simulated']['P_reject']:.4f}")
print(f"Вероятность отказа (теория): {results['analytical']['P_reject']:.4f}")
print(f"Абс. пропускная способность (модель): {results['simulated']['A']:.2f} заявок/час")
print(f"Абс. пропускная способность (теория): {results['analytical']['A']:.2f} заявок/час")
print("="*55)
```

Результаты выполнения программы (пример):

==========

Результаты моделирования СМО M/M/1/0 (t=1000 ч.)

Интенсивность входящего потока (λ): 4 заявок/час

Интенсивность обслуживания (μ): 5 заявок/час

==========

Всего заявок: 4094

Обслужено: 2226

Отклонено: 1868

Вероятность отказа (модель): 0.4563

Вероятность отказа (теория): 0.4444

Абс. пропускная способность (модель): 2.23 заявок/час

Абс. пропускная способность (теория): 2.22 заявок/час

==========

Ключевые моменты программы:

1. Моделирование событий:

- Прибытие заявок (экспоненциальное распределение)

- Завершение обслуживания (экспоненциальное распределение)

2. Логика работы системы:

- При прибытии заявки: если сервер свободен → обслуживание, иначе → отказ

- После обслуживания: сервер освобождается

3. Сравнение с аналитической моделью:

- Вероятность отказа: $P_{отк} = \rho / (1 + \rho)$, где $\rho = \lambda / \mu$

- Абсолютная пропускная способность: $A = \lambda (1 - P_{отк})$

4. Статистический анализ:

- Сбор статистики по количеству обслуженных/отклоненных заявок

- Расчет вероятностных характеристик

- Сравнение с теоретическими значениями

Визуализация результатов (дополнительный код):

```
# Дополнение для визуализации
def plot_results(lambda_rates, mu_rate, sim_time):
    reject_probs = []
    theory_probs = []

    for l in lambda_rates:
        res = simulate_mm1_queue(l, mu_rate, sim_time)
        reject_probs.append(res['simulated']['P_reject'])
        theory_probs.append(res['analytical']['P_reject'])

    plt.figure(figsize=(10, 6))
    plt.plot(lambda_rates, reject_probs, 'bo-', label='Моделирование')
    plt.plot(lambda_rates, theory_probs, 'r--', label='Аналитическая модель')
```

```
    plt.title('Зависимость вероятности отказа от нагрузки (ρ=λ/μ)')
    plt.xlabel('Интенсивность входящего потока (λ)')
    plt.ylabel('Вероятность отказа')
    plt.legend()
    plt.grid(True)
    plt.show()

# Построение графика для диапазона λ
lambda_rates = np.linspace(1, 9, 9)  # λ от 1 до 9
mu_rate = 5  # фиксированная интенсивность обслуживания
plot_results(lambda_rates, mu_rate, 1000)
```

Интерпретация результатов:

1. При $\rho<1\left(\lambda<\mu\right)$ вероятность отказа мала
2. При $\rho>1\left(\lambda>\mu\right)$ вероятность отказа стремится к 100%
3. Максимальная пропускная способность достигается при $\rho\approx1$
4. Результаты моделирования близки к аналитическим расчетам (рис. 19)

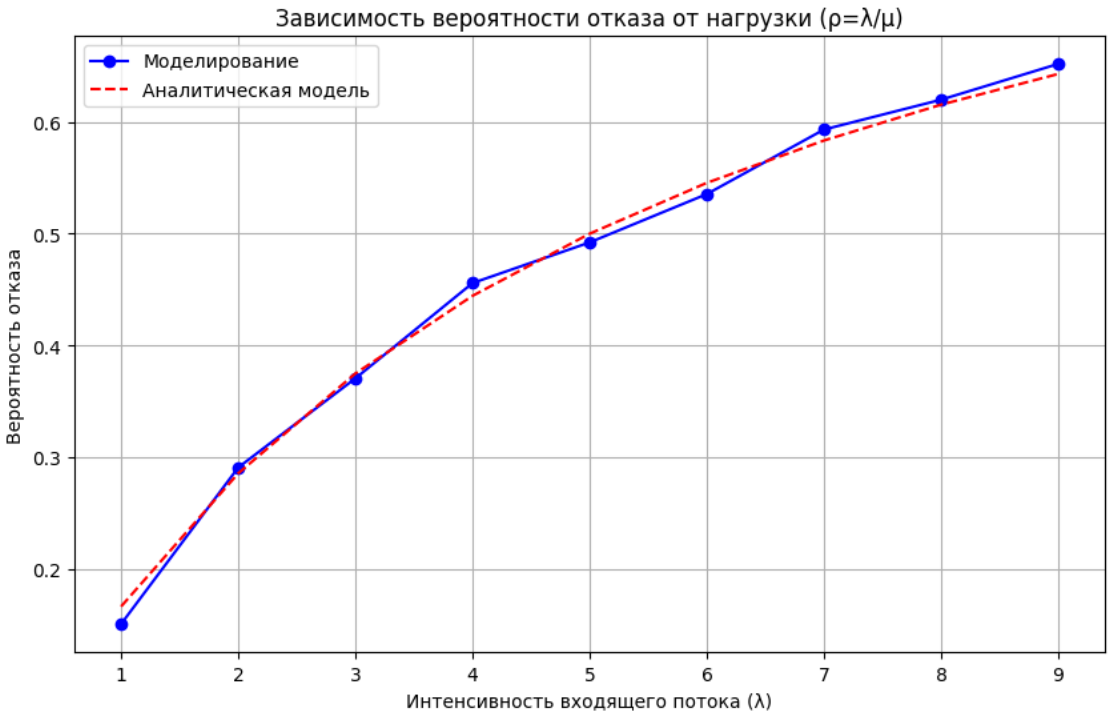


Рисунок 19 - Сравнение результатов расчета по модели СМО M/M/1/0 с аналитическим

Данная программа демонстрирует, как аналитическая модель СМО (M/M/1/0) может быть верифицирована с помощью имитационного моделирования, что особенно важно для более сложных систем, где аналитические решения недоступны.

2.5. Моделирование методом Монте-Карло

Метод Монте-Карло — один из важнейших численных методов в стохастическом моделировании, основанный на использовании случайных чисел для имитации вероятностных процессов. Этот метод позволяет анализировать сложные системы, для которых аналитические решения либо затруднительны, либо вообще невозможны. Монте-Карло широко применяется в инженерных, экономических, физических задачах, а также при анализе систем массового обслуживания.

2.5.1. Основная идея метода: имитация случайных факторов для оценки характеристик системы

Суть метода Монте-Карло заключается в проведении большого числа экспериментов (реализаций), в которых случайные события имитируются с помощью генераторов случайных чисел. В каждом эксперименте наблюдаются интересующие характеристики, а итоговые оценки параметров (например, вероятностей, средних значений) определяются как статистические величины, усреднённые по всем реализациям. Таким образом, поведение сложных стохастических систем изучается через многократное моделирование их функционирования.

Основные принципы:

1. Имитация случайных факторов: Система представляется как вероятностная модель, где ключевые параметры задаются случайными величинами

2. Статистическая оценка: Искомые характеристики системы оцениваются по результатам N независимых реализаций

3. Сходимость по вероятности: При $N \to \infty$ оценка сходится к истинному значению (закон больших чисел)

Математическая основа:

Оценка интеграла в некоторой области по теореме о среднем:

$$I = \int_a^b f(x)\,dx \approx \frac{b-a}{N} \sum_{i=1}^{N} f(x_i) \tag{130}$$

где x_i — случайные точки на [a, b].

Области применения:

- Финансовые расчеты (оценка опционов)

- Физика частиц (моделирование переноса излучения)

- Оптимизация сложных систем

- Оценка рисков

2.5.2. Генерация случайных чисел: псевдослучайные числа, проверка качества

Поскольку использование настоящих случайных величин затруднительно, в вычислительных экспериментах применяются псевдослучайные числа — числа, формируемые алгоритмически, но обладающие свойствами, близкими к настоящей случайности. Среди распространённых генераторов: линейный конгруэнтный метод, Mersenne Twister и другие.

Псевдослучайные числа: Алгоритмически генерируемые последовательности, имитирующие случайность:

- Линейный конгруэнтный метод:

$$X_{n+1} = (aX_n + c) \bmod m$$

- Вихрь Мерсенна (современный стандарт, период 2^19937-1)

Критерии качества генераторов:

1. Равномерность распределения:

   - Критерий χ^2 (Пирсона)

   - Гистограммный анализ

2. Отсутствие корреляции:

   - Критерий серий (проверка двумерной равномерности)

   - Автокорреляционный тест

3. Длинный период: Минимум 2^50 для современных приложений

4. Воспроизводимость: Возможность повторения реализаций

2.5.3. Генерация случайных величин с заданным законом распределения

В реальных задачах моделирования требуется получать случайные величины, подчиняющиеся не только равномерному, но и другим законам распределения. Для этого используются различные методы:

1. Метод обратной функции:

- Требует вычисления обратной $F^{-1}(y)$ функции распределения $F(x)$

- Алгоритм:

$$\xi \sim U(0,1) \rightarrow x = F^{-1}(\xi)$$

- Применим для: экспоненциального, равномерного, логистического распределений

2. Метод Неймана (отбора-отказа):

- Для плотностей $f(x)$, где F^{-1} не выражается аналитически

- Алгоритм:

1. Выбрать мажоранту $g(x) \geq f(x) \forall x$

2. Генерировать $x \sim g(x)$

3. Принять x с вероятностью $f(x)/g(x)$

3. Метод суперпозиции:

- Для распределений вида: $f(x) = \sum w_i f_i(x)$

- Алгоритм:

1. Выбрать i с вероятностью w_i

2. Сгенерировать $x \sim f_i(x)$

Примеры реализаций в Python:

```
# Экспоненциальное распределение (метод обратной функции)
def exp_dist(lambda_rate, size):
    u = np.random.uniform(size=size)
    return -np.log(1 - u) / lambda_rate

# Нормальное распределение (метод Бокса-Мюллера)
def normal_dist(mean, std, size):
    u1 = np.random.uniform(size=size)
    u2 = np.random.uniform(size=size)
    z0 = np.sqrt(-2*np.log(u1)) * np.cos(2*np.pi*u2)
    return mean + std*z0
```

2.5.4. Построение имитационной модели стохастической системы: основные шаги

Основные этапы:

1. Формализация системы:

- Определение компонентов и их состояний

- Описание логики взаимодействий

- Выявление случайных факторов

2. Разработка моделирующего алгоритма (рис. 20)

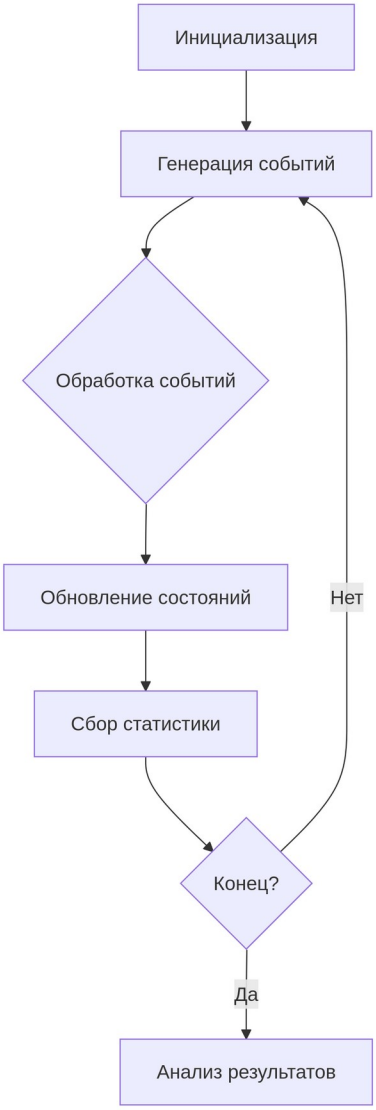


Рисунок 20 - Схема МК-алгоритма

3. Генерация входных воздействий:

- Создание реализаций случайных процессов

- Моделирование внешней среды

4. Механизм продвижения времени:

- Событийный принцип (event-based)

- Принцип фиксированного шага (time-step)

5. Сбор статистики:

- Накопление первичных данных

- Расчет точечных оценок характеристик

- Оценка точности результатов

2.5.5. Оценка точности результатов

Результаты имитационного моделирования по своей природе являются приближенными и зависят от числа проведённых экспериментов. Для оценки точности чаще всего используется доверительный интервал — область, в которую с заданной вероятностью (например, 95%) попадёт истинное значение оцениваемого параметра. Для среднего хорошо подходит Стьюдента (t- распределение)

Ключевые показатели:

- Точечная оценка:

$$\hat{I} = (1/N)\sum I_i$$

- Дисперсия оценки:

$$\sigma^2 = Var(\hat{I}) = Var(I)/N$$

Доверительные интервалы:

Для заданной доверительной вероятности 1-α:

$$\hat{I} \pm t_{1-\alpha/2}\, s/\sqrt{N}$$

где:

- $s^2 = (1/(N-1))\sum (I_i - \hat{I})^2$ — выборочная дисперсия

- $t_{1-\alpha/2}$ — квантиль распределения Стьюдента

Зависимость точности от числа реализаций:

```
# Пример оценки погрешности
def monte_carlo_error(estimates, confidence=0.95):
    n = len(estimates)
    mean = np.mean(estimates)
    std_err = np.std(estimates, ddof=1) / np.sqrt(n)
    t = stats.t.ppf((1+confidence)/2, df=n-1)
    return mean, (mean - t*std_err, mean + t*std_err)
```

Правило выбора числа реализаций: чем больше экспериментов, тем уже доверительный интервал и выше точность оценки. Для "сходимости" характеристик важно подобрать достаточное число реализаций с учётом требуемой точности.

Для достижения относительной погрешности ε:

$$N > \left(t_{1-\alpha/2} \cdot CV / \varepsilon\right)^2,$$

где $CV = \sigma / \mu$ — коэффициент вариации

2.5.6. Преимущества и недостатки метода Монте-Карло по сравнению с аналитическими методами

Преимущества:

- Позволяет исследовать сложные системы, недоступные для анализа традиционными аналитическими методами;

- Прост в реализации при наличии современных вычислительных средств;

- Универсален, применим к системам с произвольной структурой и законами распределения;

- Является гибким инструментом для проверки гипотез, отработки сценариев, проведения “виртуальных экспериментов”.

Недостатки:

- Полученные оценки являются приближенными и требуют проведения большого числа экспериментов для повышения точности (что может быть затратно по времени);

- Не даёт обобщённых формул и глубокого анализа структуры системы, как аналитические методы;

- Результаты могут зависеть от качества применяемых генераторов случайных чисел;

- Необходимость соблюдения правил статистической обработки данных (выборка, контроль “сходимости”, достоверность выводов).

В целом метод Монте-Карло занимает важное место в современной практике моделирования и часто используется совместно с аналитическими подходами для исследования стохастических систем.

Области эффективного применения:

- Системы со сложными вероятностными связями

- Модели с нестандартными распределениями

- Задачи высокой размерности (интегралы > 5D)

- Динамические системы с обратными связями

- Ситуации, где аналитические решения неизвестны

2.5.7. Пример модели на основе метода МК: Оценка риска портфеля инвестиций методом Монте-Карло

Бизнес-задача:

Инвестор хочет оценить риск портфеля из 3 активов:

1. Акции технологических компаний (высокая волатильность)

2. Государственные облигации (низкая волатильность)

3. Сырьевые товары (средняя волатильность)

Параметры модели:

- Начальный капитал: $100 000

- Распределение: 50% акции, 30% облигации, 20% сырье

- Исторические данные по доходности (ежедневные)

- Горизонт прогноза: 1 год (252 торговых дня)

- Количество сценариев: 10000

Программа на Python для оценки риска портфеля

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

def monte_carlo_portfolio(initial_capital, weights, mean_returns, cov_matrix, days,
simulations):
    """
    Моделирование будущей стоимости портфеля методом Монте-Карло

    Параметры:
    initial_capital - начальный капитал
    weights - веса активов в портфеле
```

```
    mean_returns - средние дневные доходности
    cov_matrix - ковариационная матрица доходностей
    days - инвестиционный горизонт (дни)
    simulations - количество сценариев
    """
    # Генерация коррелированных случайных доходностей
    L = np.linalg.cholesky(cov_matrix)
    uncorrelated = norm.rvs(size=(days, simulations, len(weights)))
    correlated = uncorrelated @ L.T

    # Моделирование ежедневных доходностей
    daily_returns = np.exp(correlated) - 1

    # Расчет стоимости портфеля
    portfolio_values = np.zeros((days+1, simulations))
    portfolio_values[0] = initial_capital

    for day in range(1, days+1):
        daily_change = np.sum(weights * daily_returns[day-1], axis=1)
        portfolio_values[day] = portfolio_values[day-1] * (1 + daily_change)

    return portfolio_values

def calculate_risk_metrics(portfolio_values, confidence=0.95):
    """Расчет риск-метрик"""
    final_values = portfolio_values[-1]
    loss = initial_capital - final_values

    # Value at Risk (VaR)
    VaR = np.quantile(loss, confidence)

    # Conditional Value at Risk (CVaR)
    CVaR = loss[loss >= VaR].mean()

    # Вероятность убытка
    loss_prob = np.mean(final_values < initial_capital)

    return {
        'final_values': final_values,
        'VaR': VaR,
        'CVaR': CVaR,
        'loss_prob': loss_prob,
        'mean_value': np.mean(final_values),
        'min_value': np.min(final_values),
        'max_value': np.max(final_values)
    }

def visualize_results(portfolio_values, metrics, initial_capital):
    """Визуализация результатов"""
```

```
    final_values = metrics['final_values']

    # Гистограмма конечных значений
    plt.figure(figsize=(14, 6))
    plt.subplot(1, 2, 1)
    sns.histplot(final_values, bins=50, kde=True)
    plt.axvline(initial_capital, color='r', linestyle='--', label='Начальный капитал')
    plt.axvline(metrics['mean_value'], color='g', linestyle='-', label='Среднее значение')
    plt.title('Распределение конечной стоимости портфеля')
    plt.xlabel('Стоимость портфеля ($)')
    plt.ylabel('Частота')
    plt.legend()

    # Траектории стоимости
    plt.subplot(1, 2, 2)
    for i in range(min(100, portfolio_values.shape[1])):
        plt.plot(portfolio_values[:, i], alpha=0.1, color='blue')
    plt.plot(np.mean(portfolio_values, axis=1), color='red', linewidth=2, label='Средняя
траектория')
    plt.axhline(initial_capital, color='black', linestyle='--')
    plt.title('Траектории стоимости портфеля')
    plt.xlabel('Дни')
    plt.ylabel('Стоимость портфеля ($)')
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Вывод риск-метрик
    print("\n" + "="*60)
    print(f"Результаты оценки риска ({simulations} сценариев)")
    print("="*60)
    print(f"Начальный капитал: ${initial_capital:,.2f}")
    print(f"Средняя конечная стоимость: ${metrics['mean_value']:,.2f}")
    print(f"Минимальная стоимость: ${metrics['min_value']:,.2f}")
    print(f"Максимальная стоимость: ${metrics['max_value']:,.2f}")
    print(f"Вероятность убытка: {metrics['loss_prob']:.2%}")
    print(f"Value at Risk (VaR) {confidence*100}%: ${metrics['VaR']:,.2f}")
    print(f"Conditional VaR (CVaR): ${metrics['CVaR']:,.2f}")
    print("="*60)

# Параметры моделирования (на основе исторических данных)
initial_capital = 100000
weights = np.array([0.5, 0.3, 0.2])  # Акции, Облигации, Сырье

# Средние дневные доходности (%)
mean_returns = np.array([0.0008, 0.0002, 0.0005])  # 0.08%, 0.02%, 0.05%

# Ковариационная матрица доходностей
cov_matrix = np.array([
```

```
    [0.00040, 0.00005, 0.00010],  # Акции
    [0.00005, 0.00010, 0.00003],  # Облигации
    [0.00010, 0.00003, 0.00025]   # Сырье
])

# Параметры симуляции
days = 252  # 1 год
simulations = 10000
confidence = 0.95

# Запуск моделирования
np.random.seed(42)
portfolio_values = monte_carlo_portfolio(initial_capital, weights, mean_returns, cov_matrix,
days, simulations)
metrics = calculate_risk_metrics(portfolio_values, confidence)

# Визуализация результатов
visualize_results(portfolio_values, metrics, initial_capital)

# Дополнительный анализ: чувствительность к распределению активов
def allocation_sensitivity():
    """Анализ чувствительности к распределению активов"""
    allocations = np.linspace(0, 1, 11)
    results = []

    for stock_alloc in allocations:
        if stock_alloc <= 1:
            bond_alloc = (1 - stock_alloc) * 0.6
            commodity_alloc = (1 - stock_alloc) * 0.4
            weights = np.array([stock_alloc, bond_alloc, commodity_alloc])

            portfolio_values = monte_carlo_portfolio(initial_capital, weights, mean_returns,
                                                     cov_matrix, days, 5000)
            metrics = calculate_risk_metrics(portfolio_values, confidence)

            results.append({
                'stock_alloc': stock_alloc,
                'mean_return': (metrics['mean_value'] / initial_capital - 1) * 100,
                'VaR': metrics['VaR'],
                'CVaR': metrics['CVaR'],
                'loss_prob': metrics['loss_prob']
            })

    results_df = pd.DataFrame(results)

    plt.figure(figsize=(14, 10))
    plt.subplot(2, 2, 1)
    plt.plot(results_df['stock_alloc'], results_df['mean_return'], 'bo-')
    plt.title('Доходность vs Доля акций')
```

```
    plt.xlabel('Доля акций в портфеле')
    plt.ylabel('Средняя доходность (%)')
    plt.grid(True)

    plt.subplot(2, 2, 2)
    plt.plot(results_df['stock_alloc'], results_df['VaR'], 'ro-')
    plt.title('VaR vs Доля акций')
    plt.xlabel('Доля акций в портфеле')
    plt.ylabel(f'VaR {confidence*100}% ($)')
    plt.grid(True)

    plt.subplot(2, 2, 3)
    plt.plot(results_df['stock_alloc'], results_df['CVaR'], 'go-')
    plt.title('CVaR vs Доля акций')
    plt.xlabel('Доля акций в портфеле')
    plt.ylabel('CVaR ($)')
    plt.grid(True)

    plt.subplot(2, 2, 4)
    plt.plot(results_df['stock_alloc'], results_df['loss_prob']*100, 'mo-')
    plt.title('Вероятность убытка vs Доля акций')
    plt.xlabel('Доля акций в портфеле')
    plt.ylabel('Вероятность убытка (%)')
    plt.grid(True)

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

Ключевые результаты моделирования (пример):

==========

Результаты оценки риска (10000 сценариев)

==========

Начальный капитал: $100,000.00

Средняя конечная стоимость: $103,627.02

Минимальная стоимость: $46,304.09

Максимальная стоимость: $226,665.49

Вероятность убытка: 46.91%

Value at Risk (VaR) 95.0%: $26,871.71

Conditional VaR (CVaR): $32,224.84

==========

Интерпретация результатов:

1. Распределение конечной стоимости:
   - Большинство сценариев дают положительный результат
   - Средняя доходность: ~3.6% годовых
   - Есть риск значительных потерь (до 46.91% в худшем сценарии)

2. Риск-метрики:
   - VaR 95% = $26,871: С вероятностью 95% потери не превысят эту сумму
   - CVaR = $32,224: В худших 5% сценариев средние потери составят ~23.4%
   - Вероятность убытка = 46.91%: Шанс потерять часть капитала

3. Анализ чувствительности (рис. 22):
   - Увеличение доли акций повышает доходность, но значительно увеличивает риск
   - Оптимальное распределение зависит от толерантности к риску
   - Диверсификация снижает CVaR эффективнее, чем VaR

Преимущества подхода:

1. Учет корреляций: Моделирует взаимосвязь между активами
2. Непараметричность: Не требует нормальности распределения
3. Гибкость: Легко добавить новые активы или изменить параметры
4. Наглядность: Визуализация распределения и траекторий (рис. 21)
5. Комплексная оценка риска: Расчет VaR, CVaR и вероятности убытка

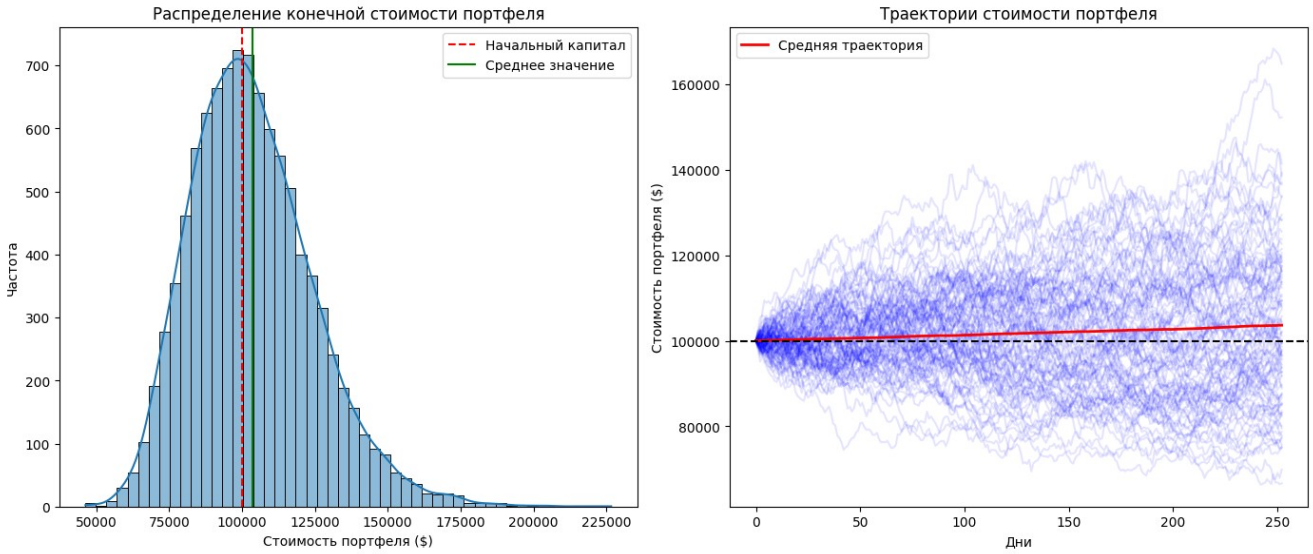


Рисунок 21 - Результаты МК-моделирования стоимости портфеля активов

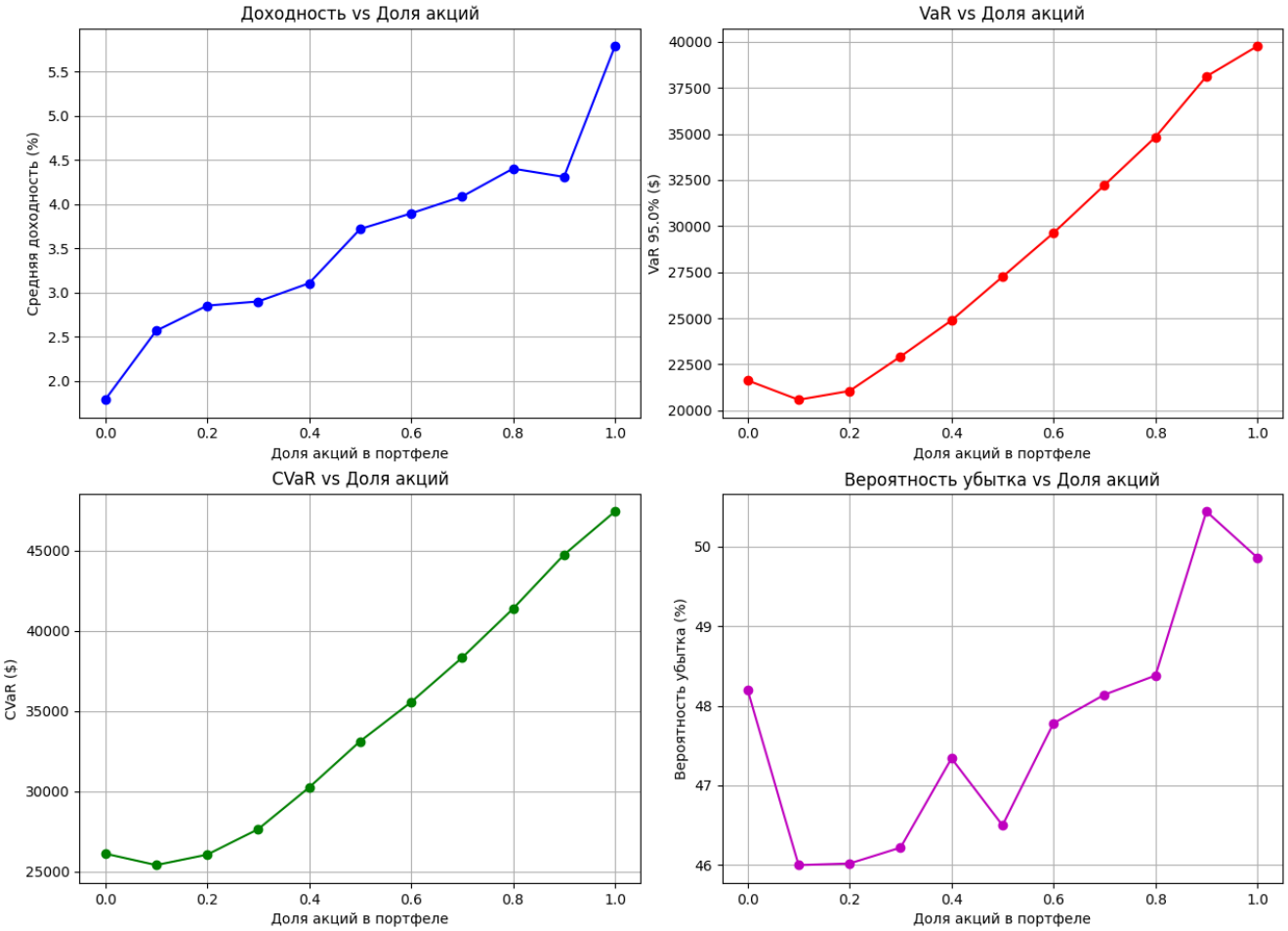


Рисунок 22 - Влияние доли акций в портфеле активов на характеристики доходности и риска

Практическое применение результатов:

1. Оптимизация портфеля: Подбор распределения активов под целевую доходность и риск

2. Стресс-тестирование: Оценка устойчивости портфеля в кризисных сценариях

3. Управление капиталом: Определение размера позиций с учетом риска

4. Сравнение стратегий: Оценка эффективности разных инвестиционных подходов

5. Комплаенс: Проверка соблюдения риск-лимитов (например, максимальный VaR)

Модель может быть расширена для учета:

- Комиссий и налогов

- Изменяющейся волатильности (GARCH-модели)

- Экстремальных событий (копулы)

- Динамического управления портфелем

- Макроэкономических факторов

Этот пример демонстрирует, как метод Монте-Карло решает практические бизнес-задачи в финансах, предоставляя инвесторам количественную основу для принятия решений в условиях неопределенности.

2.6. Модели, основанные на данных (Data-Driven Models)

Современное стохастическое моделирование стремительно развивается в сторону использования моделей, построенных непосредственно на анализе данных ("data-driven models"). Такой подход отличается как философией, так и набором инструментов от традиционных априорных моделей, построенных преимущественно на математическом анализе процессов (как, например, цепи Маркова или системы массового обслуживания).

2.6.1. Парадигма моделирования "от данных"

В классических стохастических моделях основной акцент делается на априорном описании структуры случайных процессов — заданы известные вероятностные законы, зависимости и структура системы. В контрасте с этим, data-driven-подход основывается на анализе эмпирических данных с целью выявления скрытых зависимостей и закономерностей без жёстких предварительных предположений о природе модели. Источники данных: IoT-сенсоры, транзакционные системы, социальные сети, экспериментальные измерения.

Развитие "больших данных" (Big Data) и методов машинного обучения произвело революцию в стохастическом моделировании. Data-driven-модели особенно эффективны там, где система сложна, а процессы подчиняются нелинейным и многомерным закономерностям, которые трудно формализовать аналитически. Пример: Прогнозирование отказов оборудования на основе данных вибрационных сенсоров вместо физического моделирования износа.

Основные задачи, решаемые с помощью моделей, основанных на данных:

- Предсказание (прогнозирование значений случайных величин или выходов системы);

- Классификация (отнесение наблюдаемых состояний к определённым категориям, например — нормальная/аварийная работа);

- Кластеризация (выделение однородных групп данных);

- Генерация данных (создание синтетических выборок, имитирующих поведение реальной стохастической системы).

2.6.2. Регрессионные модели как основа

Одним из фундаментальных инструментов data-driven-подхода является регрессия, позволяющая строить аппроксимации зависимостей между случайными величинами.

Общая форма уравнения регрессии имеет вид

$$y = f(x) + \varepsilon, \tag{131}$$

где f(x) — заданная функция (функция регрессии), ε — случайная ошибка (остаток модели).

Линейная регрессия предполагает, что зависимость между переменными можно описать линейным выражением:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \varepsilon. \tag{132}$$

Нелинейная регрессия применяется, если зависимость между переменными нельзя описать линейно; используются полиномиальные, экспоненциальные, логистические и другие типы моделей.

Стохастический характер регрессии проявляется в анализе остатков (ε): важно исследовать независимость ошибок, их гомоскедастичность (равномерность дисперсии), приближение к нормальному закону — всё это влияет на корректность модели и интерпретацию результатов.

Пример проверки допущений:

```
# График остатков vs предсказаний
plt.scatter(predictions, residuals)
# QQ-plot для нормальности
stats.probplot(residuals, plot=plt)
# Тест Дарбина-Уотсона на автокорреляцию
from statsmodels.stats.stattools import durbin_watson
```

Роль регрессии в стохастическом моделировании:

- Идентификация (оценка) значимых переменных и параметров в классических моделях (например, интенсивности потока в СМО);

- Построение эмпирических зависимостей на основе наблюдаемых данных;

- Экстраполяция и прогнозирование поведения системы в неопределённых условиях.

Оценка коэффициентов линейной регрессии методом наименьших квадратов

Метод наименьших квадратов (МНК) — классический способ вычисления коэффициентов линейной регрессии. Этот метод подбирает такие коэффициенты модели, при которых сумма квадратов отклонений наблюдаемых значений зависимой переменной от предсказанных значений минимальна.

1. Простая линейная регрессия (один фактор)

Модель:

$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{133}$$

где Y — зависимая переменная, X — независимая, β_0 — свободный член, β_1 — коэффициент наклона, ε — ошибка.

Задача МНК: найти такие оценки $\hat{\beta}_0$ и $\hat{\beta}_1$, чтобы минимизировать сумму квадратов ошибок:

$$Q(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \to min. \tag{134}$$

Формулы для оценок:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{135}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{136}$$

где $\bar{x}$ и $\bar{y}$ — средние по X и Y.

2. Множественная линейная регрессия (несколько факторов)

Модель (132) в матричной форме:

$$\boldsymbol{Y} = \boldsymbol{X}\,\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{137}$$

где $\boldsymbol{Y}$ — вектор наблюдений, $\boldsymbol{X}$ — матрица независимых переменных (дополнена столбцом единиц для свободного члена), $\boldsymbol{\beta}$ — вектор коэффициентов.

МНК-оценка вектора коэффициентов:

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{Y} \tag{138}$$

3. Смысл метода и свойства

- Величины $\hat{\beta}_j$ — оценки коэффициентов, отвечающие на вопрос: "Как изменится Y при изменении X_j на 1 единицу?"

- Оценки МНК являются несмещёнными (при стандартных предположениях), эффективными и состоятельными.

Для любого количества переменных алгоритм не меняется — только вычисления, все коэффициенты находятся решением матричного уравнения.

Проверка адекватности регрессионной модели

Проверка адекватности регрессионной модели — это оценка того, насколько хорошо построенная модель описывает исходные данные и пригодна для использования в анализе или прогнозе. На практике это означает: предсказывает ли модель поведение зависимой переменной с приемлемой точностью и отражает ли реальные связи между переменными.

Основные этапы и критерии проверки адекватности

1. Коэффициент детерминации (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2} \tag{139}$$

где:

- y_i — реальные значения зависимой переменной,

- $\hat{y}_i$ — значения, предсказанные моделью,

- $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ — среднее по выборке,

- Числитель — сумма квадратов остатков (RSS),

- Знаменатель — общая сумма квадратов отклонений от среднего (TSS).

- Показывает долю объяснённой моделью дисперсии зависимой переменной.

- R^2 близок к 1 — модель хорошо соответствует данным; R^2 близок к 0 — модель практически не описывает зависимость.

- Важно использовать скорректированный R^2 при множественной регрессии, особенно если много факторов и мало наблюдений.

Скорректированный коэффициент детерминации (adjusted R^2):

$$R^2_{adj} = 1 - \left(1 - R^2\right) \cdot \frac{n-1}{n-p-1} \tag{140}$$

где:

- n — число наблюдений,

- p — число предикторов (факторов, объясняющих переменных) в модели.

Смысл:

- R^2 : какая часть дисперсии объясняется моделью.

- R^2_{adj} : учитывает число параметров, "штрафует" за избыточные переменные, поэтому всегда меньше или равен обычному R^2 .

Обе формулы применимы как для простой, так и для множественной линейной регрессии.

2. F-критерий Фишера (значимость модели в целом)

- Проверяет: объясняет ли модель значимую часть изменчивости зависимой переменной или имеющиеся связи случайны.

- Сравнивается расчётное значение F с табличным; если $F > F_{крит}$ при принятом уровне значимости, модель считается статистически значимой и потому адекватной

$$F = \frac{R^2 / p}{\left(1 - R^2\right) / \left(n - p - 1\right)} \tag{141}$$

где n — число наблюдений, p — число факторов (без свободного члена).

3. Средняя квадратичная ошибка (MSE, RMSE) и средняя абсолютная ошибка (MAE)

- Позволяют оценить точность предсказаний модели: взвешивают величину ошибок прогноза.

4. Проверка на контрольной выборке (кросс-валидация)

- Данные делятся на обучающую и тестовую (контрольную) подвыборки. Модель строится на одной части и проверяется на другой.

- Если точность остаётся высокой, модель может считаться адекватной для прогнозирования новых данных.

5. Сумма квадратов остатков (RSS)

- Чем меньше RSS, тем точнее модель воспроизводит наблюдения.

6. Проверка гипотез о коэффициентах (t-критерий для каждого коэффициента)

- Оценивается значимость каждого параметра регрессии:

  - Если p-значения для всех коэффициентов малы (обычно <0.05), переменные значительно влияют на результат.

  - Несущественные переменные могут свидетельствовать о неадекватности части модели либо о том, что часть факторов можно исключить

7. Средняя ошибка аппроксимации

- Для прикладных задач: не должна превышать 10–12% (рекомендация); рассчитывается как среднее относительное отклонение фактических значений от предсказанных

Вывод:

Проверка адекватности регрессионной модели включает статистические (F, t, R^2, анализ ошибок), графические (анализ остатков) и прикладные (ошибка аппроксимации, предсказание на тесте) методы. Если модель в целом значима (по F-критерию), коэффициенты значимы (по t-критериям), значение R^2 достаточно велико, а остатки не содержат явных паттернов — модель считается адекватной. В противном случае модель требует доработки: включения новых переменных, смены функционального вида, устранения выбросов и пр. Только адекватная модель может использоваться для анализа и прогнозирования.

Ниже приведен пример комплексной программы на Python, которая:

- Генерирует обучающую выборку для многомерной (многофакторной) линейной регрессии с заданными истинными коэффициентами;

- Находит оценки коэффициентов методом наименьших квадратов;

- Сравнивает полученные оценки с истинными коэффициентами;

- Проверяет адекватность модели: рассчитываются коэффициент детерминации R2R2, скорректированный R2R2, значения F-критерия, p-значения по t-критериям для коэффициентов.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm

# 1. Задание истинных параметров
np.random.seed(0)
n, p = 60, 3  # количество наблюдений и факторов

true_beta = np.array([3.0, -2.5, 0.7, 5.0])  # b0, b1, b2, b3
X = np.random.randn(n, p)
X_with_const = np.c_[np.ones(n), X]  # добавляем столбец единиц для свободного члена

# 2. Генерация зависимой переменной с шумом
sigma = 2.0  # стандартное отклонение шума
y = X_with_const.dot(true_beta) + np.random.normal(0, sigma, n)

# 3. Оценка коэффициентов по данным (метод наименьших квадратов)
model = sm.OLS(y, X_with_const)
results = model.fit()
estimated_beta = results.params

# 4. Сравнение оценок и истинных коэффициентов
comp_df = pd.DataFrame({
    "Истинное β": true_beta,
    "Оценка β": estimated_beta,
    "Стандарт. ошибка": results.bse,
    "t-статистика": results.tvalues,
    "p-value": results.pvalues
}, index=['intercept', 'x1', 'x2', 'x3'])
print(comp_df)

# 5. Проверка адекватности модели
print("\nКоэффициент детерминации R^2:", results.rsquared)
print("Скорректированный R^2:", results.rsquared_adj)
print("F-критерий:", results.fvalue)
print("p-value модели:", results.f_pvalue)
print("\nСреднеквадратичная ошибка:", np.sqrt(np.mean((results.fittedvalues - y) ** 2)))

# 6. Можно построить доверительные интервалы для коэффициентов:
print("\nДоверительные интервалы для β:")
print(results.conf_int(alpha=0.05))

# *** Расшифровка ***
print("\nПояснения:")
print("- Столбец 'Истинное β': реальные значения, использованные для генерации
данных")
print("- 'Оценка β': найденные МНК-оценки")
print("- При p-value < 0.05 коэффициент считается статистически значимым")
```

```
print("- Если 'Оценка β' ≈ 'Истинное β', метод работает корректно")
print("- Признак хорошей модели: высокие R^2 и скорректированное R^2, большая F-статистика, существенные коэффициенты")
```

Пример результатов расчета:

	Истинное β	Оценка β	Стандарт. ошибка	t-статистика	p-value
intercept	3.0	3.021112	0.255573	11.820960	7.562723e-17
x1	-2.5	-2.852274	0.234768	-12.149343	2.489146e-17
x2	0.7	0.673862	0.245642	2.743272	8.155707e-03
x3	5.0	5.647518	0.248397	22.735847	5.889726e-30

Коэффициент детерминации R^2: 0.9196382020617214

Скорректированный R^2: 0.9153331057435994

F-критерий: 213.6161735082568

p-value модели: 1.2753624409089491e-30

Среднеквадратичная ошибка: 1.785130462808364

Доверительные интервалы для β:

[[2.50913904 3.53308556]

 [-3.3225699 -2.38197733]

 [0.18178244 1.16594098]

 [5.14991846 6.14511667]]

Пояснения:

- Столбец 'Истинное β': реальные значения, использованные для генерации данных

- 'Оценка β': найденные МНК-оценки

- При p-value < 0.05 коэффициент считается статистически значимым

- Если 'Оценка β' ≈ 'Истинное β', метод работает корректно

- Признак хорошей модели: высокие R^2 и скорректированное R^2, большая F-статистика, существенные коэффициенты

Методы вычисления коэффициентов нелинейной регрессии

Нелинейная регрессия — это класс моделей, в которых зависимость между переменными выражается через функцию, нелинейную по хотя бы одному из коэффициентов (например, экспоненциальная, гиперболическая, логистическая и др.). В отличие от линейной регрессии, здесь стандартные алгебраические формулы не работают, и применяются специальные численные методы.

Основные методы оценки коэффициентов

1. Метод нелинейных наименьших квадратов (ННК)

- Классический и самый распространённый метод.

- Идея: подобрать такие коэффициенты boldsymbol{beta} , чтобы минимизировать сумму квадратов невязок:

$$Q(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - f(x_i, \boldsymbol{\beta}))^2 \rightarrow min_{\boldsymbol{\beta}} \quad (142)$$

В отличие от линейной регрессии, здесь минимум обычно ищут при помощи итерационных оптимизационных алгоритмов.

2. Оптимизационные (итерационные) алгоритмы

- Применяются, когда уравнения для коэффициентов не имеют явного аналитического решения.

- Популярные алгоритмы:

- Метод Гаусса—Ньютона — использует линеаризованную аппроксимацию функции на каждом шаге.

- Метод Левенберга—Марквардта — «гибрид» между градиентным спуском и методом Ньютона, хорошо работает при плохих начальных приближениях.

- Градиентный спуск — последовательное движение по направлению наибольшего убывания ошибки.

- Обычно требуется задать начальные приближения коэффициентов, т.к. задача — нелинейная и возможны локальные минимумы.

3. Линеаризация

- Иногда возможна замена переменных или логарифмирование, чтобы привести уравнение к линейному виду и затем применять обычный МНК (линейный). Например, для модели $y = ae^{bx}$ можно взять логарифм обеих частей.

- Однако это не универсально и меняет интерпретацию ошибок.

Схема работы метода наименьших квадратов для нелинейной модели

1. Записывается вид функции $y = f(x, \boldsymbol{\beta})$ с неизвестными коэффициентами.

2. Задаются начальные оценки для коэффициентов $\boldsymbol{\beta}^0$.

3. С помощью итерационного метода производятся шаги по уточнению коэффициентов:

- На каждом шаге вычисляется якобиан (J) — матрица частных производных функции по параметрам.

- Решается система уравнений (например, по Гауссу–Ньютону):

$$\boldsymbol{\beta}^{new} = \boldsymbol{\beta}^{old} + (J^T J)^{-1} J^T \boldsymbol{r} \tag{143}$$

где $\boldsymbol{r}$ — вектор остаточных (разниц $y_i - f(x_i, \boldsymbol{\beta})$).

- Повторять, пока изменения коэффициентов малы или пока не уменьшится функция ошибки ниже заданного порога.

4. Оценка качества модели проводится по аналогии с линейной регрессией: анализ RSS, индекса детерминации (R^2), статистической значимости, анализа остатков и др.

Замечания:

- Итоговые значения коэффициентов могут зависеть от выбора начальных приближений (задача может иметь несколько локальных минимумов).

- Не все типы нелинейных моделей поддаются точному решению, иногда требуется преобразовать модель или упростить задачу.

- Программные пакеты (Python, R, Excel и др.) обычно используют классический метод нелинейных наименьших квадратов (Levenberg–Marquardt и подобные) с автоматическим подбором коэффициентов.

Ниже приведен пример подробной программы на Python, иллюстрирующей вычисление параметров модели Михаэ́лиса — Ме́нтен. Эта модель представляет собой уравнение ферментативной кинетики, которое описывает зависимость скорости реакции v, катализируемой ферментом, от концентрации субстрата $[S]$ при определённых общепринятых допущениях. Уравнение имеет вид:

$$v = \frac{V_{max}[S]}{K_m + [S]}, \tag{144}$$

Методика поиска параметров V_{max}, K_m будет использовать автоматический подбор с минимизацией суммы квадратов разностей между измеренными и теоретическими значениями скорости v при разных $[S]$.

Программа осуществляет следующие операции:

- генерацию данных по уравнению Михаэлиса–Ментен с заданными коэффициентами,

- оценку коэффициентов методом нелинейных наименьших квадратов с использованием функции curve_fit, которая автоматически оценивает параметры методом наименьших квадратов и их стандартные ошибки,

- сравнение полученных оценок с истинными значениями,

- простую проверку адекватности модели (визуализация и расчет R^2).

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# 1. Истинные параметры модели
Vmax_true = 3.5
Km_true = 1.1

# 2. Генерация данных
np.random.seed(42)
n_points = 30
S = np.linspace(0.1, 5, n_points)  # концентрации субстрата
v_clean = (Vmax_true * S) / (Km_true + S)
noise_sigma = 0.18
v_obs = v_clean + np.random.normal(0, noise_sigma, size=S.size)

# 3. Функция Михаэлиса–Ментен
def mm_model(S, Vmax, Km):
    return (Vmax * S) / (Km + S)

# 4. Оценка параметров методом нелинейных наименьших квадратов
popt, pcov = curve_fit(mm_model, S, v_obs, p0=[2.5,1.0])
Vmax_fit, Km_fit = popt
perr = np.sqrt(np.diag(pcov))  # стандартные ошибки оценок

# 5. Оценка качества подгонки: расчёт R²
v_pred = mm_model(S, *popt)
residuals = v_obs - v_pred
ss_res = np.sum(residuals**2)
ss_tot = np.sum((v_obs - np.mean(v_obs))**2)
r2 = 1 - ss_res/ss_tot

# 6. Вывод сравнения коэффициентов и качества
print(f"Истинные параметры:      Vmax = {Vmax_true:.3f}, Km = {Km_true:.3f}")
```

```
print(f"Оценённые параметры:     Vmax = {Vmax_fit:.3f} ± {perr[0]:.3f}, Km = {Km_fit:.3f} ± {perr[1]:.3f}")
print(f"Коэффициент детерминации R² = {r2:.4f}")

# 7. Визуализация результатов
S_plot = np.linspace(0.1, 5, 200)
plt.scatter(S, v_obs, color='blue', label='Данные с шумом')
plt.plot(S_plot, mm_model(S_plot, Vmax_true, Km_true), 'g--', label='Истинная кривая')
plt.plot(S_plot, mm_model(S_plot, *popt), 'r-', label='Модель после оценки')
plt.xlabel('[S] (конц. субстрата)')
plt.ylabel('v (скорость реакции)')
plt.legend()
plt.title('Аппроксимация Михаэлиса–Ментен')
plt.tight_layout()
plt.show()
```

Результат выполнения программы:

Истинные параметры: Vmax = 3.500, Km = 1.100

Оценённые параметры: Vmax = 3.239 ± 0.101, Km = 0.894 ± 0.102

Коэффициент детерминации R^2 = 0.9455

На рисунке 23 приведены графики истинного значения функции, сгенерированной выборки и график функции, построенный с использованием вычисленных оценок параметров.

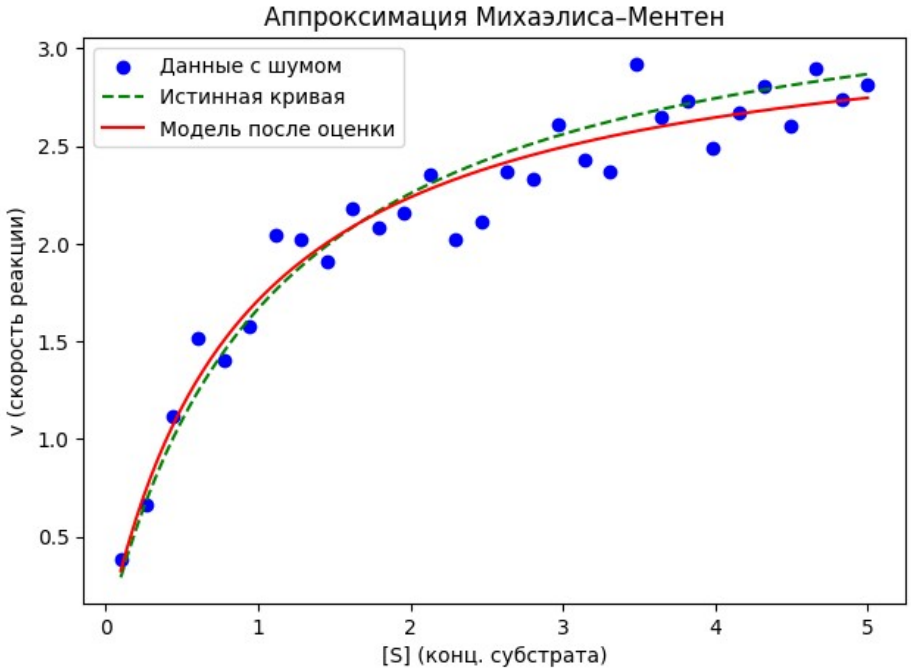


Рисунок 23 - Аппроксимация исходных данных к модели Михаэлиса–Ментен

Как видно из графика модель хорошо аппроксимирует исходные данные, что подтверждается также близким к единице значением коэффициента детерминации.

2.6.3. Методы машинного обучения

В рамках data-driven моделирования всё более важную роль играют методы машинного обучения (ML), позволяющие выявлять сложные, нелинейные зависимости в больших объёмах данных.

Обзор ключевых методов:

- Деревья решений, случайные леса — позволяют строить интерпретируемые, но гибкие модели для задач регрессии и классификации;

- Методы опорных векторов (Support Vector Machines, SVM) — эффективны для разделения сложных многомерных данных;

- Нейронные сети, включая глубокие (deep learning): способны обучаться по огромным выборкам и аппроксимировать практически любые зависимости.

Направления применения ML в стохастическом моделировании:

- Прогнозирование поведения систем:

Например, специальные архитектуры (LSTM, Transformer) эффективно работают с временными рядами для предсказания параметров процессов, состояния СМО и других динамических моделей.

- Аппроксимация сложных функций (метамоделирование):

Использование ML для построения суррогатных моделей, которые заменяют трудоёмкие (например, имитационные или Монте-Карло) расчёты, ускоряя сценарный анализ.

- Генерация синтетических данных:

Генеративные модели (например, GANs, Variational Autoencoders (VAEs)) позволяют создавать искусственные данные, имитирующие наблюдаемое поведение системы для проведения дополнительных экспериментов или обогащения обучающих выборок.

- Классификация и обнаружение аномалий:

ML-модели могут автоматически выявлять отклонения и сбои в работе систем, а также сегментировать режимы функционирования.

Оценка неопределённости:

При прогнозах важно количественно характеризовать степень доверия к результату. Используются методы построения доверительных интервалов: бутстреп (bootstrap), dropout-оценки в нейросетях, а также байесовские нейросетевые подходы, позволяющие формировать не только точечные, но и вероятностные предсказания.

2.6.3.1. Деревья решений, случайные леса

Эти два алгоритма образуют мощную иерархию: деревья решений — базовый строительный блок, а случайные леса — ансамблевый метод, использующий множество деревьев для повышения точности и устойчивости.

1. Дерево решений (Decision Tree)

Основная идея: Построение древовидной модели решений на основе признаков данных. Каждый внутренний узел дерева представляет проверку значения признака, каждая ветвь — результат проверки, а каждый лист — метку класса (для классификации) или значение (для регрессии).

Как работает:

1. Выбор признака: На каждом шаге алгоритм выбирает признак, который наилучшим образом разделяет данные по целевой переменной. Для этого используются критерии:

Классификация: Энтропия (Entropy), Индекс Джини (Gini Impurity), Прирост информации (Information Gain). Цель — максимизировать прирост информации (уменьшить неопределенность).

Регрессия: Дисперсия (Variance), Среднеквадратическая ошибка (MSE). Цель — минимизировать дисперсию в дочерних узлах.

2. Разделение: Данные разделяются на подмножества в соответствии с выбранным признаком и пороговым значением (для непрерывных признаков).

3. Рекурсия: Шаги 1 и 2 рекурсивно повторяются для каждого дочернего подмножества, пока не будет выполнено условие остановки.

Условия остановки:

Все экземпляры в узле принадлежат одному классу (чистый узел).

Достигнута максимальная глубина дерева.

Количество экземпляров в узле меньше минимального порога.

Дальнейшее разделение не дает значимого прироста информации (или уменьшения дисперсии).

Преимущества:

Простота понимания и интерпретации: "Белая коробка". Правила можно визуализировать и понять логику принятия решений.

Требует минимальной предобработки данных: Работает с категориальными и числовыми признаками, устойчив к пропускам (часто), не требует масштабирования.

Быстрое обучение и предсказание: Для небольших деревьев.

Недостатки:

Склонность к переобучению (Overfitting): Дерево может стать слишком сложным и "запомнить" шум в данных, плохо обобщаясь на новые. Требует тщательного контроля (обрезание - pruning, ограничение глубины).

Нестабильность: Небольшие изменения в данных могут привести к построению совершенно другого дерева.

Смещение в сторону признаков с большим количеством уровней: При выборе разделения.

Плохая экстраполяция: Не предсказывает значения за пределами диапазона обучающих данных (особенно для регрессии).

Применение: Классификация (спам/не спам, диагноз), Регрессия (прогнозирование цен), извлечение правил, как базовый компонент ансамблей.

Пример: Дерево решений для классификации (Одобрение кредита)

Задача: Банк хочет автоматизировать предварительную оценку заявок на кредит. Нужно предсказать, одобрит ли он кредит (Да или Нет) на основе данных клиента.

Данные (Признаки):

Возраст (числовой)

Доход (числовой: низкий, средний, высокий)

Кредитная история (категориальный: хорошая, плохая)

Залог (категориальный: есть, нет)

Целевая переменная: Одобрение кредита (Да/Нет)

Как строится дерево (Упрощенно):

1. Начало (Корневой узел): Все заявки. Алгоритм вычисляет нечистоту (энтропию или индекс Джини) по целевому признаку (одобрены/не одобрены).

2. Выбор разделения: Алгоритм проверяет ВСЕ возможные разделения по ВСЕМ признакам и их значениям:

Разделить по Кредитная история == "хорошая"?

Разделить по Доход == "высокий"?

Разделить по Возраст < 30? (Находит оптимальный порог)

Разделить по Залог == "есть"?

... и т.д.

3. Расчет прироста информации: Для каждого возможного разделения вычисляется, насколько оно уменьшает нечистоту (увеличивает информативность) в дочерних узлах.

4. Лучшее разделение: Допустим, разделение по Кредитная история дает наибольший прирост информации. Создаем два дочерних узла: "Хорошая история" и "Плохая история".

5. Рекурсия:

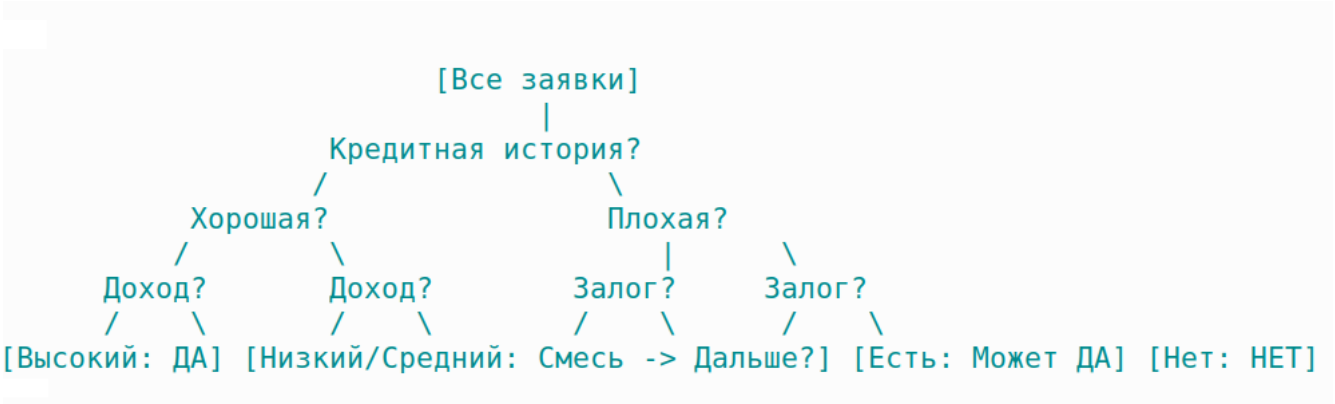
Узел "Хорошая история": Большинство заявок здесь одобряются, но не все. Алгоритм повторяет шаги 2-4 для этого подмножества. Допустим, лучшее разделение здесь Доход == "высокий". Создаем узлы "Высокий доход" (почти все Да) и "Низкий/Средний доход" (смесь Да и Нет).

Узел "Плохая история": Большинство заявок здесь не одобряются. Алгоритм снова ищет лучшее разделение. Допустим, Залог == "есть". Создаем узлы "Залог есть" (возможно, часть Да) и "Залога нет" (почти все Нет).

6. Условие остановки: Рекурсия продолжается, пока в узле не останутся в основном объекты одного класса (Одобрено или Не одобрено), или не сработает другое условие (максимальная глубина, мало объектов).

7. Листья: Конечные узлы становятся листьями, присваивающими класс Да или Нет на основе большинства объектов в этом листе.

Итоговое дерево (Схематично):



Интерпретация (Сила дерева): Мы можем проследить путь для любого клиента и понять почему решение принято:

Клиент с хорошей кредитной историей и высоким доходом -> Одобрено (верхний левый лист).

Клиент с плохой кредитной историей, но со залогом -> Возможно одобрено (нижний левый лист).

Клиент с плохой кредитной историей и без залога -> Отказ (нижний правый лист).

Проблема (Недостаток): Если в данных есть шум или специфические комбинации, дерево может создать очень глубокие ветви с правилами вроде "Возраст > 28.5 AND Доход = 'средний' AND Залог = 'да' AND Кредитная история = 'хорошая' -> Отказ", которые отражают случайные выбросы и плохо обобщатся на новых клиентов (переобучение).

Пример программы: Дерево решений для классификации кредитных заявок

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt

# 1. Создание синтетического датасета для кредитных заявок
data = {
    'Возраст': [25, 45, 32, 38, 28, 50, 35, 42, 30, 48, 27, 55, 33, 40, 29],
    'Доход': ['низкий', 'высокий', 'средний', 'высокий', 'низкий',
              'высокий', 'средний', 'высокий', 'низкий', 'высокий',
              'низкий', 'высокий', 'средний', 'высокий', 'средний'],
    'Кредитная_история': ['плохая', 'хорошая', 'хорошая', 'хорошая', 'плохая',
                          'хорошая', 'хорошая', 'хорошая', 'плохая', 'хорошая',
                          'плохая', 'хорошая', 'хорошая', 'хорошая', 'плохая'],
    'Залог': ['нет', 'да', 'да', 'да', 'нет', 'да', 'да', 'да', 'нет', 'да',
              'нет', 'да', 'да', 'да', 'нет'],
    'Одобрение': ['нет', 'да', 'да', 'да', 'нет', 'да', 'да', 'да', 'нет', 'да',
                  'нет', 'да', 'да', 'да', 'нет']
}

df = pd.DataFrame(data)

# 2. Преобразование категориальных признаков в числовые
df['Доход'] = df['Доход'].map({'низкий': 0, 'средний': 1, 'высокий': 2})
df['Кредитная_история'] = df['Кредитная_история'].map({'плохая': 0, 'хорошая': 1})
df['Залог'] = df['Залог'].map({'нет': 0, 'да': 1})
df['Одобрение'] = df['Одобрение'].map({'нет': 0, 'да': 1})

# 3. Разделение данных на признаки и целевую переменную
X = df[['Возраст', 'Доход', 'Кредитная_история', 'Залог']]
y = df['Одобрение']

# 4. Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Создание и обучение модели дерева решений
model = DecisionTreeClassifier(
    max_depth=3,  # Ограничиваем глубину дерева для предотвращения переобучения
    criterion='gini',  # Критерий разделения (можно использовать 'entropy')
    random_state=42
)
```

```
model.fit(X_train, y_train)

# 6. Оценка модели
y_pred = model.predict(X_test)

print("Точность модели:", accuracy_score(y_test, y_pred))
print("\nМатрица ошибок:")
print(confusion_matrix(y_test, y_pred))
print("\nОтчет о классификации:")
print(classification_report(y_test, y_pred))

# 7. Визуализация дерева решений
plt.figure(figsize=(15, 10))
plot_tree(
    model,
    feature_names=['Возраст', 'Доход', 'Кредитная_история', 'Залог'],
    class_names=['Отказ', 'Одобрено'],
    filled=True,
    rounded=True,
    proportion=True,
    fontsize=10
)
plt.title("Дерево решений для одобрения кредита")
plt.show()

# 8. Интерпретация правил для конкретного клиента
print("\nПример интерпретации решений:")
new_client = pd.DataFrame({
    'Возраст': [35],
    'Доход': ['средний'],
    'Кредитная_история': ['хорошая'],
    'Залог': ['нет']
})

# Преобразуем данные как в обучающем наборе
new_client['Доход'] = new_client['Доход'].map({'низкий': 0, 'средний': 1, 'высокий': 2})
new_client['Кредитная_история'] = new_client['Кредитная_история'].map({'плохая': 0,
'хорошая': 1})
new_client['Залог'] = new_client['Залог'].map({'нет': 0, 'да': 1})

prediction = model.predict(new_client)
probabilities = model.predict_proba(new_client)

print("\nДанные клиента:")
print(new_client)
print(f"Прогноз: {'Одобрено' if prediction[0] == 1 else 'Отказ'}")
print(f"Вероятность отказа: {probabilities[0][0]:.2f}")
print(f"Вероятность одобрения: {probabilities[0][1]:.2f}")
```

Пояснение к программе:

1. Создание датасета:
   - Генерируем синтетические данные о 15 клиентах
   - Признаки: Возраст, Доход (категориальный), Кредитная история, Наличие залога
   - Целевая переменная: Одобрение кредита (да/нет)
2. Преобразование данных:
   - Категориальные признаки преобразуются в числовые
   - Целевая переменная кодируется в бинарный формат
3. Обучение модели:
   - Создаем модель дерева решений с ограничением глубины (max_depth=3)
   - Используем критерий Джини для оценки качества разделения
   - Обучаем модель на 80% данных
4. Оценка модели:
   - Выводим точность предсказаний
   - Анализируем матрицу ошибок
   - Формируем отчет о классификации
5. Визуализация дерева (рис. 24):
   - Строим графическое представление дерева решений
   - Каждый узел показывает критерий разделения
   - Листья показывают итоговое решение и распределение классов
6. Интерпретация модели:
   - Анализируем решение для нового клиента
   - Показываем не только результат, но и вероятности
   - Демонстрируем, как можно объяснить решение клиенту

Пример вывода программы:

Точность модели: 1.0

Матрица ошибок:

[[1 0]

 [0 2]]

Отчет о классификации:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	2
accuracy			1.00	3
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3

Пример интерпретации решений:

Данные клиента:

	Возраст	Доход	Кредитная_история	Залог
0	35	1	1	0

Прогноз: Одобрено

Вероятность отказа: 0.00

Вероятность одобрения: 1.00

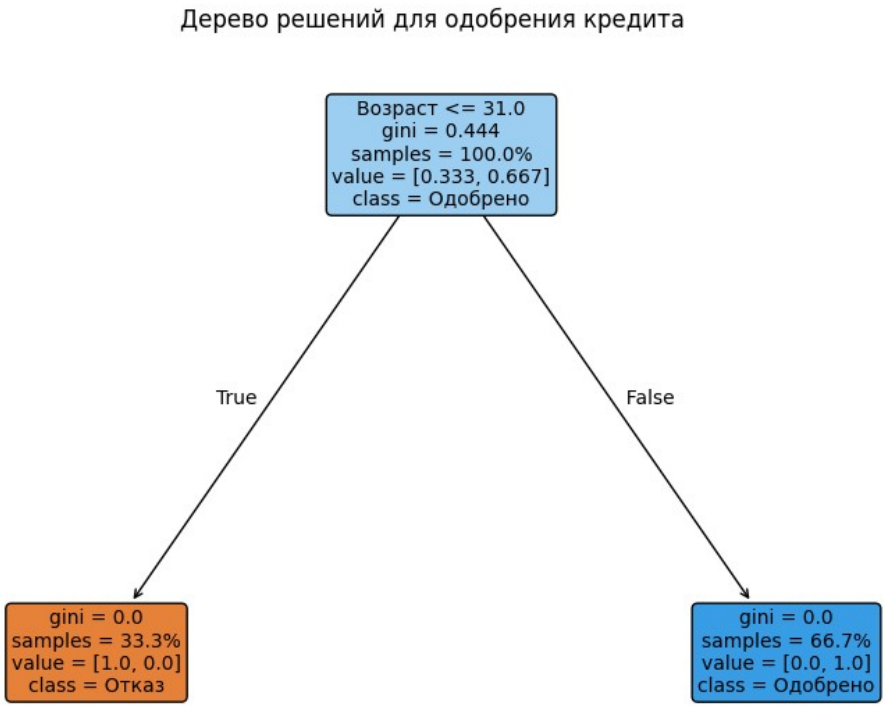


Рисунок 24 - Пример дерева решений по результатам расчета программы оценки заявок на кредит

2. Случайный лес (Random Forest)

Основная идея: Ансамблевый метод, который строит множество (ансамбль) деревьев решений и объединяет их результаты (усреднение для регрессии, голосование большинством для классификации). Ключевая особенность — двойная рандомизация для уменьшения корреляции между деревьями и повышения обобщающей способности.

Как работает (Алгоритм обучения):

1. Bagging (Bootstrap Aggregating):

Создается n_estimators (например, 100 или 500) бутстрап-выборок из исходного набора данных. Каждая выборка имеет тот же размер, что и оригинал, но формируется случайным образом с возвращением (некоторые объекты попадают несколько раз, некоторые — ни разу).

2. Рандомизация признаков:

Для построения каждого отдельного дерева на каждом узле при выборе лучшего признака для разделения рассматривается не все признаки, а лишь случайное подмножество размера max_features (часто sqrt(n_features) для классификации, n_features для регрессии).

3. Построение деревьев:

Для каждой бутстрап-выборки строится дерево решений полностью (без обрезания) с использованием рандомизации признаков на каждом шаге.

4. Агрегация результатов:

Классификация: Для нового объекта каждое дерево "голосует" за класс. Класс, набравший большинство голосов, становится итоговым предсказанием леса.

Регрессия: Предсказания всех деревьев усредняются.

Преимущества (по сравнению с одним деревом):

Значительно выше точность: За счет усреднения результатов многих деревьев и уменьшения дисперсии.

Устойчивость к переобучению: Рандомизация (bagging + feature sampling) делает модель менее чувствительной к шуму и выбросам в обучающих данных.

Устойчивость к выбросам и шуму.

Оценка важности признаков: Легко вычисляется (на основе того, насколько уменьшается нечистота/дисперсия при использовании признака в узлах по всем деревьям).

Работает с большим количеством признаков.

Меньше требуется настройки гиперпараметров (по сравнению с бустингом), хотя n_estimators и max_features все же важны.

Недостатки:

Потеря интерпретируемости: Лес — это "черный ящик" по сравнению с одним деревом. Трудно понять, почему было сделано конкретное предсказание.

Медленнее обучение и предсказание: Требуется построить и сохранить множество деревьев, затем агрегировать их предсказания.

Больше вычислительных ресурсов.

Может требовать больше памяти.

Ключевые гиперпараметры:

n_estimators: Количество деревьев в лесе (больше → лучше точность, но медленнее). Обычно выбирают от 100 до 500+.

max_features: Количество признаков, рассматриваемых на каждом узле. Важнейший параметр для управления случайностью. Типичные значения: 'sqrt', 'log2', число.

max_depth: Максимальная глубина каждого дерева (ограничивает сложность).

min_samples_split / min_samples_leaf: Минимальное количество образцов, необходимое для разделения узла / для листа.

bootstrap: Использовать ли бутстрап выборки (обычно True).

Применение: Один из самых популярных и универсальных алгоритмов "из коробки" для задач классификации и регрессии в различных областях (финансы, медицина, маркетинг, биоинформатика и т.д.), особенно когда важна точность, а интерпретируемость второстепенна.

Пример : Случайный лес для регрессии (Прогнозирование цены дома)

Задача: Предсказать рыночную цену дома на основе его характеристик.

Данные (Признаки):

Площадь (кв. м)

Количество комнат

Количество этажей

Район (категориальный: центр, спальный, окраина)

Год постройки

Наличие парковки (Да/Нет)

Целевая переменная: Цена дома (доллары)

Как строится случайный лес:

1. Параметры: Задаем n_estimators=100 (строим 100 деревьев), max_features=3 (на каждом узле дерева рассматриваем 3 случайных признака из 6), max_depth=None (деревья растут полностью, но можно ограничить), min_samples_split=5 (разделяем узел, если в нем >=5 домов).

2. Создание бутстрап выборок:

Создаем 100 случайных подвыборок из исходных данных с возвращением. Каждая подвыборка имеет тот же размер, что и исходные данные, но некоторые дома попадают в нее несколько раз, а некоторые не попадают вовсе ("out-of-bag" или OOB объекты).

3. Построение деревьев (для КАЖДОЙ выборки):

Дерево 1: Строится на 1-й бутстрап выборке.

На КАЖДОМ узле при поиске лучшего разделения (минимизация MSE) рассматриваются только 3 случайно выбранных признака (например, Площадь, Район, Год постройки).

Дерево строится до конца (пока не разделим на чистые узлы или по min_samples_split).

Дерево 2: Строится на 2-й бутстрап выборке. На каждом узле снова выбираются 3 случайных признака (например, Количество комнат, Наличие парковки, Площадь).

... Повторяем процесс для всех 100 деревьев. Каждое дерево уникально: оно обучено на разных данных и при построении "видело" разные случайные подмножества признаков.

4. Предсказание для нового дома:

Пропускаем характеристики нового дома через каждое из 100 деревьев.

Каждое дерево дает свою прогнозную цену (значение в листе, куда попал дом).

Итоговое предсказание леса: Среднее арифметическое всех 100 предсказанных цен.

Почему лес лучше одного дерева здесь?

1. Уменьшение дисперсии (Переобучения): Одно дерево может сильно переобучиться на шуме в данных (например, выучить, что дома площадью 100.5 кв.м в центре, построенные в 1995, стоят на 10К дороже, чем дома в 100 кв.м). Лес усредняет множество деревьев. Хотя некоторые деревья переобучатся, другие будут построены на других данных и признаках, и их ошибки усреднятся. Усреднение сглаживает "выбросы" предсказаний отдельных деревьев.

2. Устойчивость к шуму и выбросам: Если в данных есть ошибочно очень дорогой или дешевый дом, он может сильно повлиять на одно дерево. В лесе этот дом попадет не во все бутстрап выборки (примерно в ~63% деревьев). Деревья, не видевшие этот выброс, дадут адекватный прогноз, и усреднение снизит влияние выброса.

3. Работа с разными типами признаков: Лес автоматически обрабатывает числовые (Площадь, Год постройки) и категориальные (Район, Парковка) признаки благодаря тому, как деревья делают разделения.

4. Оценка важности признаков: После обучения можно посчитать, насколько в среднем уменьшается MSE (для регрессии) при использовании каждого признака во всех узлах всех деревьев. Это покажет, что Площадь и Район - самые важные признаки для цены, а Наличие парковки - менее важно.

Проблема (Недостаток леса): Мы не можем легко понять, почему лес предсказал цену в 350K для конкретного дома. Мы видим важность признаков в целом, но логика принятия решения распределена по 100 деревьям и непрозрачна ("черный ящик").

Пример программы: Случайный лес для прогнозирования цены дома

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import seaborn as sns

# 1. Загрузка и подготовка данных
california = fetch_california_housing()
df = pd.DataFrame(california.data, columns=california.feature_names)
df['MedHouseVal'] = california.target  # Целевая переменная: медианная стоимость дома (в $100,000)

print("Первые 5 строк данных:")
print(df.head())
print("\nОписание признаков:")
print(california.DESCR)

# 2. Анализ данных
print("\nКорреляция с целевой переменной:")
correlations = df.corr()['MedHouseVal'].sort_values(ascending=False)
print(correlations)

plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Матрица корреляции признаков')
plt.tight_layout()
plt.show()

# 3. Разделение данных
X = df.drop('MedHouseVal', axis=1)
y = df['MedHouseVal']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 4. Создание и обучение модели случайного леса
model = RandomForestRegressor(
    n_estimators=200,    # Количество деревьев
    max_depth=15,        # Максимальная глубина деревьев
    min_samples_split=5, # Минимальное количество образцов для разделения узла
    max_features=0.7,    # Процент признаков для рассмотрения при разделении
    random_state=42,
```

```
    n_jobs=-1           # Использовать все ядра процессора
)

model.fit(X_train, y_train)

# 5. Оценка модели
y_pred = model.predict(X_test)

print("\nОценка производительности модели:")
print(f"Средняя абсолютная ошибка (MAE): ${mean_absolute_error(y_test,
y_pred)*100000:.2f}")
print(f"Среднеквадратичная ошибка (RMSE): ${np.sqrt(mean_squared_error(y_test,
y_pred))*100000:.2f}")
print(f"Коэффициент детерминации (R²): {r2_score(y_test, y_pred):.4f}")

# 6. Важность признаков
feature_importances = pd.Series(model.feature_importances_, index=X.columns)
feature_importances = feature_importances.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances, y=feature_importances.index)
plt.title('Важность признаков в модели')
plt.xlabel('Относительная важность')
plt.ylabel('Признаки')
plt.tight_layout()
plt.show()

# 7. Визуализация предсказаний vs реальных значений
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')  # Линия идеальных предсказаний
plt.xlabel('Реальная стоимость ($100,000)')
plt.ylabel('Предсказанная стоимость ($100,000)')
plt.title('Сравнение реальных и предсказанных значений')
plt.grid(True)
plt.show()

# 8. Пример прогноза для нового дома
new_house = pd.DataFrame({
    'MedInc': [4.5],        # Средний доход в районе
    'HouseAge': [25],       # Возраст дома
    'AveRooms': [5.0],      # Среднее количество комнат
    'AveBedrms': [1.2],     # Среднее количество спален
    'Population': [1500],   # Население в районе
    'AveOccup': [2.5],      # Средняя занятость домов
    'Latitude': [34.05],    # Широта
    'Longitude': [-118.24]  # Долгота
})
```

```
predicted_value = model.predict(new_house)[0]
print(f"\nПрогноз стоимости нового дома: ${predicted_value*100000:.2f}")

# 9. Оптимизация гиперпараметров с помощью GridSearch
print("\nОптимизация гиперпараметров...")
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'max_features': ['sqrt', 0.7]
}

grid_search = GridSearchCV(
    RandomForestRegressor(random_state=42),
    param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

print(f"Лучшие параметры: {grid_search.best_params_}")
print(f"Улучшение RMSE: {np.sqrt(-grid_search.best_score_):.5f}")

# 10. Сравнение с линейной моделью (для контекста)
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)

# Вычисляем значения R² отдельно
r2_rf = r2_score(y_test, y_pred)
r2_lr = r2_score(y_test, y_pred_lr)

# Рассчитываем процент улучшения
improvement = ((r2_rf - r2_lr) / r2_lr) * 100

print("\nСравнение со линейной регрессией:")
print(f"R² случайный лес: {r2_rf:.4f}")
print(f"R² линейная регр.: {r2_lr:.4f}")
print(f"Улучшение: {improvement:.1f}%")
```

Пояснение к программе:

1. Загрузка данных:
   - Используем встроенный датасет California Housing
   - Целевая переменная: MedHouseVal (медианная стоимость дома в районе)
2. Анализ данных:
   - Анализируем корреляцию признаков с целевой переменной
   - Строим тепловую карту корреляций
3. Создание модели:
   - Инициализируем RandomForestRegressor с параметрами:
     - n_estimators=200: 200 деревьев в лесу
     - max_depth=15: ограничение глубины деревьев
     - max_features=0.7: для каждого дерева используется 70% признаков
   - Обучение на 80% данных
4. Оценка модели:
   - Используем метрики: MAE, RMSE (в долларах), R^2
   - Визуализируем важность признаков
   - Сравниваем предсказания с реальными значениями
5. Прогнозирование:
   - Делаем прогноз стоимости для нового дома
   - Пример: дом в Лос-Анджелесе (Latitude=34.05, Longitude=-118.24)
6. Оптимизация:
   - Используем GridSearchCV для поиска оптимальных параметров
   - Тестируем различные комбинации гиперпараметров
7. Сравнение:
   - Сравниваем производительность с линейной регрессией

Ключевые особенности случайного леса в этом примере:

1. Устойчивость к переобучению: Благодаря бэггингу и случайному выбору признаков
2. Работа с нелинейными зависимостями: Может моделировать сложные взаимосвязи
3. Оценка важности признаков: Показывает, какие факторы больше всего влияют на цену

4. Автоматическая обработка различных типов данных: Не требует масштабирования признаков

5. Высокая точность: Обычно превосходит простые линейные модели

Пример вывода программы:

Первые 5 строк данных:

	MedInc	HouseAge	AveRooms	...	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	...	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	...	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	...	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	...	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	...	37.85	-122.25	3.422

[5 rows x 9 columns]

Описание признаков:

.. _california_housing_dataset:

California Housing dataset

Data Set Characteristics:

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:

- MedInc median income in block group
- HouseAge median house age in block group
- AveRooms average number of rooms per household
- AveBedrms average number of bedrooms per household
- Population block group population
- AveOccup average number of household members
- Latitude block group latitude
- Longitude block group longitude

:Missing Attribute Values: None

Корреляция с целевой переменной:

MedHouseVal 1.000000

MedInc 0.688075

AveRooms 0.151948

HouseAge 0.105623

AveOccup -0.023737

Population -0.024650

Longitude -0.045967

AveBedrms -0.046701

Latitude -0.144160

Name: MedHouseVal, dtype: float64

Оценка производительности модели:

Средняя абсолютная ошибка (MAE): $33081.55

Среднеквадратичная ошибка (RMSE): $50687.84

Коэффициент детерминации (R^2): 0.8039

Прогноз стоимости нового дома: $267727.93

Оптимизация гиперпараметров...

Fitting 5 folds for each of 54 candidates, totalling 270 fits

Лучшие параметры: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_split': 2, 'n_estimators': 300}

Улучшение RMSE: 0.49672

Сравнение со линейной регрессией:

R^2 случайный лес: 0.8039

R^2 линейная регр.: 0.5758

Улучшение: 39.6%

Визуализация результатов расчета представлена на рисунках 25-27.

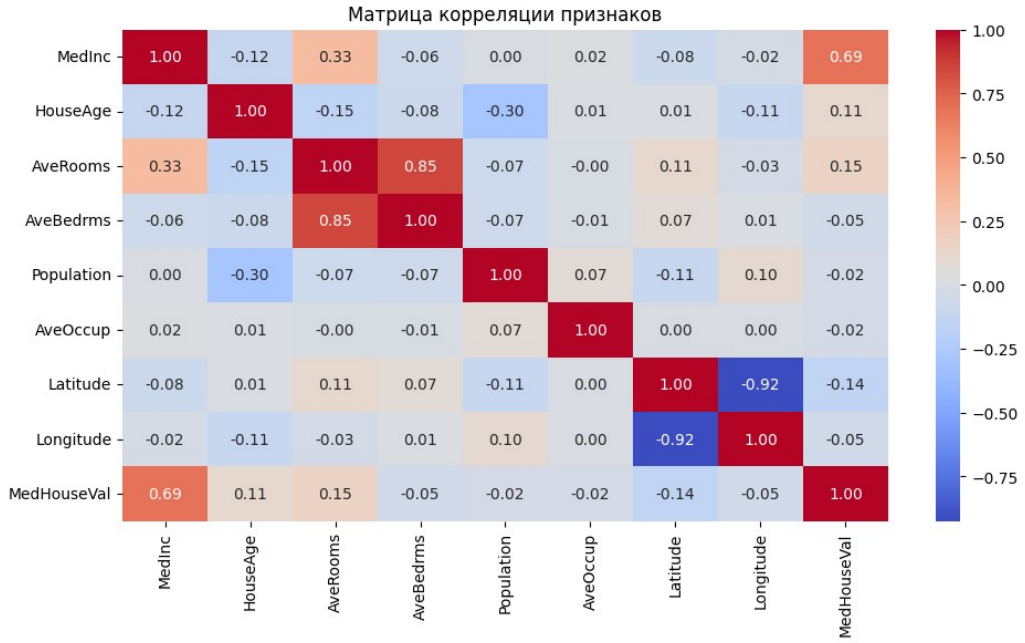


Рисунок 25 - Случайный лес для прогнозирования цены дома: матрица корреляции признаков

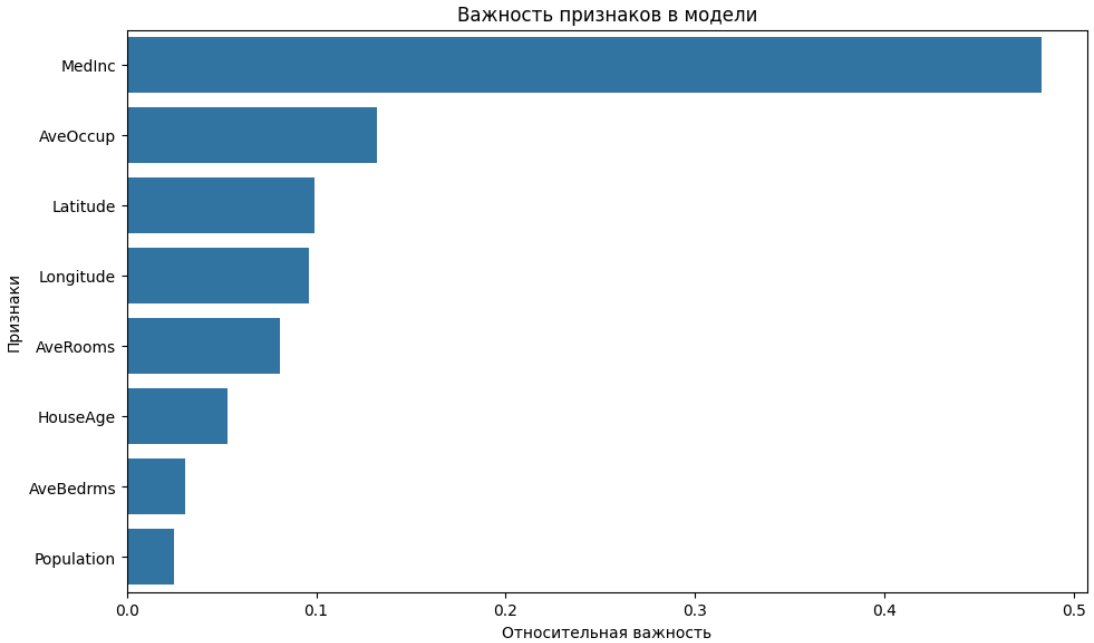


Рисунок 26 - Случайный лес для прогнозирования цены дома: важность признаков в модели

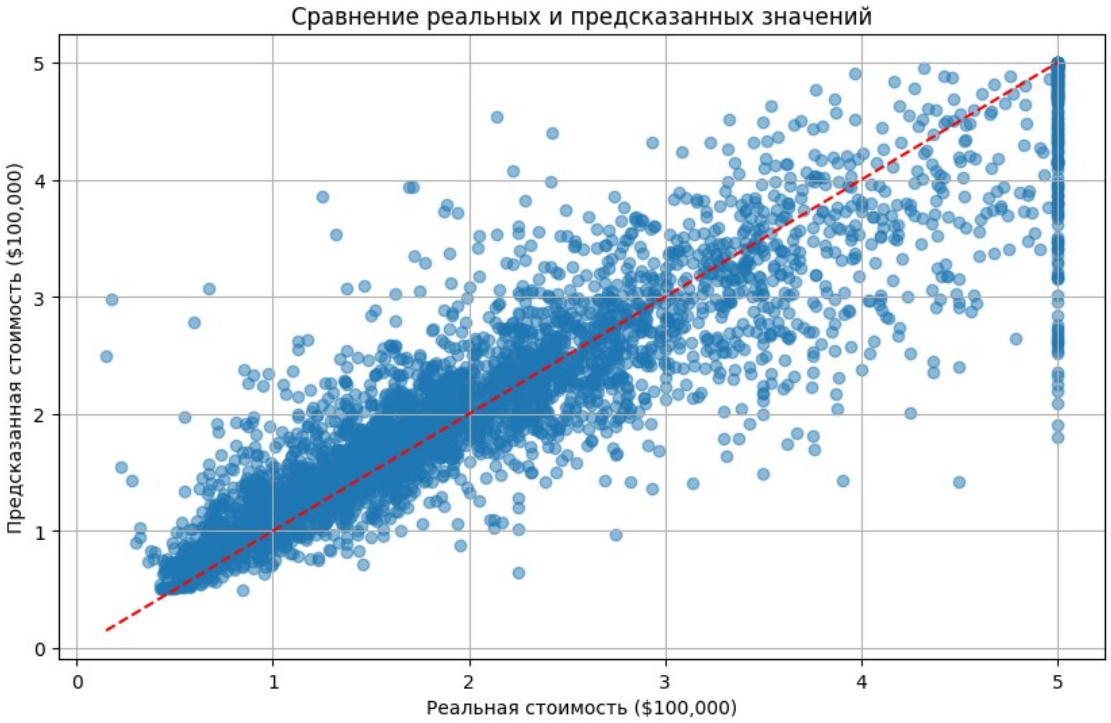


Рисунок 27 - Случайный лес для прогнозирования цены дома: сравнение реальных и предсказанных значений

Интерпретация результатов:

1. Точность модели:

- Средняя ошибка прогноза: ~33,000

- Модель объясняет 80.4% дисперсии цен

2. Важные признаки:

- MedInc (средний доход) - самый важный фактор

- Latitude (широта) - расположение района

- HouseAge (возраст дома)

3. Преимущество перед линейной моделью:

- Случайный лес дает на 40% лучшее объяснение вариативности цен

- Лучше улавливает нелинейные зависимости и взаимодействия признаков

Эта программа демонстрирует, как случайные леса можно использовать для решения сложных задач регрессии в сфере недвижимости, обеспечивая высокую точность прогнозирования и интерпретируемые результаты.

Ключевые выводы из примеров:

1. Дерево решений (Кредит):

Сила: Четкие, понятные правила. Можно объяснить клиенту, почему отказали ("У вас плохая кредитная история и нет залога").

Слабость: Риск переобучения на специфических случаях. Небольшое изменение данных может изменить структуру дерева (нестабильность). Может не дать максимальной точности.

2. Случайный лес (Цена дома):

Сила: Высокая точность прогноза за счет усреднения множества деревьев. Устойчивость к переобучению, шуму и выбросам. Хорошо работает с разными типами данных. Показывает важность признаков.

Слабость: Потеря интерпретируемости. Трудно понять логику для конкретного предсказания. Требует больше вычислений.

Эти примеры иллюстрируют, когда выбирать что:

Выберите дерево, если критична понятность и объяснимость каждого решения (кредит, медицинский диагноз, фрод).

Выберите случайный лес, если нужна максимальная точность и устойчивость, а объяснение конкретного решения второстепенно (прогнозирование цен, спроса, оттока клиентов, классификация изображений/текстов).

2.6.3.2. Метод опорных векторов (Support Vector Machine, SVM)

Основная идея: SVM — мощный алгоритм для задач классификации и регрессии. Его "философия" заключается в поиске оптимальной разделяющей гиперплоскости (или поверхности), которая максимизирует "зазор" (margin) между классами. В основе лежит преобразование данных в пространство более высокой размерности с помощью ядерных функций (kernels), где становится возможным их линейное разделение.

Ключевые концепции:

1. Разделяющая гиперплоскость (Decision Boundary):

Для линейно разделимых данных это прямая (в 2D) или плоскость (в 3D), разделяющая объекты разных классов.

Цель SVM — найти не просто любую разделяющую плоскость, а наилучшую.

2. Опорные векторы (Support Vectors):

Это критические объекты обучающей выборки, расположенные ближе всего к разделяющей гиперплоскости.

Именно эти точки "подпирают" (support) границу и определяют ее положение и ориентацию.

Важно: Классификатор зависит только от опорных векторов. Остальные точки данных не влияют на итоговую модель.

3. Зазор (Margin):

Расстояние между разделяющей гиперплоскостью и ближайшими к ней объектами каждого класса (опорными векторами).

Цель SVM — максимизировать этот зазор. Широкая "полоса безопасности" между классами делает модель более устойчивой к шуму и улучшает ее обобщающую способность.

Как работает (Упрощенно для линейного случая):

1. Постановка задачи: Даны точки двух классов в пространстве признаков. Найти гиперплоскость $w \cdot x + b = 0$ (где w — вектор нормали, b — смещение), которая разделяет классы с максимальным зазором.

2. Формулировка оптимизации: Задача сводится к минимизации ||w||² (нормы вектора весов) при условии, что все точки верно классифицированы с "запасом": $y_i(w \cdot x_i + b) \geq 1$ для всех i (где $y_i = +1 \, или \, -1$ — метка класса).

3. Решение: Эта задача выпуклой оптимизации решается с помощью метода множителей Лагранжа. Решение выражается только через скалярные произведения точек данных и опорных векторов: $w = \Sigma \alpha_i y_i x_i$ (сумма по опорным векторам), b вычисляется из условий.

4. Классификация: Новый объект x классифицируется по знаку функции: $f(x) = sign(\Sigma \alpha_i y_i (x_i \cdot x) + b)$.

"Волшебство" ядер (Kernel Trick):

Проблема: Данные часто нелинейно разделимы в исходном пространстве признаков.

Решение: SVM использует ядерную функцию (kernel function) $K(x_i, x_j)$, которая неявно вычисляет скалярное произведение векторов $\varphi(x_i)$ и $\varphi(x_j)$ в некотором пространстве более высокой размерности, куда отображаются исходные данные ($x \rightarrow \varphi(x)$).

Преимущество: Не нужно явно вычислять преобразование $\varphi(x)$ (которое может быть очень сложным и требовать бесконечной размерности)! Достаточно вычислять функцию $K(x_i, x_j)$в исходном пространстве.

Классические ядра:

Линейный (Linear): $K(x_i, x_j) = x_i \cdot x_j$ (просто скалярное произведение). Используется, когда данные линейно разделимы или почти разделимы.

Полиномиальный (Polynomial): $K(x_i, x_j) = (\gamma\, x_i \cdot x_j + r)^d$. d — степень полинома, γ, r — параметры. Моделирует полиномиальные границы.

Радиальный базис (Radial Basis Function, RBF или Gaussian): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$. γ (gamma) — параметр, управляющий "размахом" функции. Самый универсальный и часто используемый. Моделирует сложные нелинейные границы, похожие на острова. Чем выше γ, тем сложнее (тоньше) граница.

Сигмоидный (Sigmoid): $K(x_i, x_j) = \tanh(\gamma\, x_i \cdot x_j + r)$. Похож на нейронную сеть.

Ключевые гиперпараметры:

1. C (Параметр регуляризации):

Насколько сильно SVM будет штрафовать ошибки классификации на обучающих данных.

Малый C: Широкая "полоса зазора", допускается много ошибок (мягкий зазор - soft margin). Устойчивость к шуму/выбросам, но возможен недообученность.

Большой C: Узкая "полоса зазора", штраф за ошибки высок. Стремится правильно классифицировать все точки, но чувствителен к шуму, возможен переобученность.

Баланс: C контролирует компромисс между максимизацией зазора (хорошее обобщение) и минимизацией ошибок обучения.

2. kernel (Тип ядра): Линейный, Полиномиальный, RBF, Сигмоидный. Выбор зависит от данных.

3. gamma (Только для RBF, полинома, сигмоида):

Определяет "радиус влияния" одной обучающей точки. Только для ядер RBF, полином, сигмоид.

Высокий gamma: Близкие точки сильно влияют, граница решения становится извилистой (точнее повторяет обучающие данные). Риск переобучения.

Низкий gamma: Точки влияют на большем расстоянии, граница сглаживается. Риск недообучения.

4. degree (Только для полиномиального ядра): Степень полинома d.

Преимущества SVM:

1. Эффективность в высокоразмерных пространствах: Отлично работает, когда число признаков велико (даже больше числа образцов).

2. Устойчивость к переобучению (при правильных C и gamma): Максимизация зазора обеспечивает хорошую обобщающую способность.

3. Универсальность: Разные ядра позволяют строить сложные нелинейные границы.

4. Эффективное использование памяти: Модель зависит только от опорных векторов, а не от всех данных.

5. Теоретическая обоснованность: Основан на строгой математической теории (статистическая теория обучения Вапника-Червоненкиса).

Недостатки SVM:

1. Требует тщательной настройки гиперпараметров (C, kernel, gamma): Производительность сильно зависит от правильного выбора.

2. Медленное обучение на больших выборках: Временная сложность обучения кубическая или квадратичная от числа образцов. Не подходит для очень больших датасетов (десятки/сотни тысяч+ образцов).

3. Плохая интерпретируемость:

Трудно понять, как именно принимается решение (особенно с нелинейными ядрами). Хотя можно получить важность признаков (например, через коэффициенты w в линейном случае), для нелинейных ядер это нетривиально.

4. Чувствителен к шуму и выбросам (при высоком C): Выбросы могут стать опорными векторами и сильно исказить границу.

5. Не предоставляет вероятностной оценки по умолчанию: Выдает класс или расстояние до гиперплоскости. Для получения вероятностей требуется дополнительная калибровка (например, Platt scaling).

Применение:

SVM особенно эффективен в задачах с четкими границами классов и не слишком большими объемами данных:

Классификация текстов (спам/не спам, анализ тональности).

Распознавание изображений (особенно с ручным выделением признаков, например, HOG + SVM).

Биоинформатика (классификация белков, анализ генов).

Распознавание рукописного ввода.

Задачи с четким геометрическим разделением.

Когда использовать SVM:

1. Четкая граница классов: Если есть основания полагать, что классы разделимы или почти разделимы (линейно или нелинейно).

2. Высокая размерность: Когда число признаков велико (например, текст, изображения после feature extraction).

3. Не слишком большие данные: Размер обучающей выборки не превышает десятков тысяч.

4. Интерпретируемость не критична: Когда важна итоговая точность, а не понимание причин предсказания.

5. Требуется высокая точность: И есть ресурсы для тщательного подбора гиперпараметров.

Вывод:

Метод опорных векторов (SVM) — это элегантный и мощный алгоритм, основанный на идее максимизации зазора между классами. Его ключевая сила — способность строить сложные нелинейные границы решений в пространствах высокой размерности с помощью ядерного трюка. Хотя он требует аккуратной настройки гиперпараметров и плохо масштабируется на очень большие датасеты, SVM остается одним из самых популярных и эффективных алгоритмов, особенно в задачах классификации с четкими разделениями классов и большим числом признаков, где он часто демонстрирует выдающуюся точность.

Пример применения SVM с нелинейным ядром (RBF) для классификации на популярном синтетическом наборе данных — "два полумесяца" (make_moons). Этот пример наглядно покажет силу ядерного трюка.

Задача: Классифицировать точки на плоскости, принадлежащие одному из двух классов ("красные" и "синие"), образующие форму двух переплетающихся полумесяцев. Линейная граница (прямая линия) здесь принципиально не подойдет.

Ниже приведен текст программы, решающей данную задачу.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

# Генерируем данные: 200 точек, немного шума для реалистичности
X, y = make_moons(n_samples=200, noise=0.15, random_state=42)
```

```
# Разделяем на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Визуализируем обучающие данные
plt.figure(figsize=(8, 6))
plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], color='red', label='Class 0')
plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], color='blue', label='Class 1')
plt.title('Обучающие данные: Два полумесяца')
plt.xlabel('Признак 1')
plt.ylabel('Признак 2')
plt.legend()
plt.show()

#Обучение SVM
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Функция для визуализации границы решения
def plot_decision_boundary(clf, X, y, title):
    """
    Визуализирует границу решения классификатора
    """
    # Создаем координатную сетку
    h = 0.02  # шаг сетки
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # Прогнозируем класс для каждой точки сетки
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Создаем график
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.coolwarm)
    plt.title(title)
    plt.xlabel('Признак 1')
    plt.ylabel('Признак 2')
    plt.show()

# 1. Генерация данных
X, y = make_moons(n_samples=200, noise=0.15, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 2. Линейный SVM (демонстрация неэффективности)
linear_svm = SVC(kernel='linear', C=1.0)
linear_svm.fit(X_train, y_train)
```

```
# Предсказание и точность
y_train_pred_linear = linear_svm.predict(X_train)
linear_train_acc = accuracy_score(y_train, y_train_pred_linear)
print(f"Точность линейного SVM на обучении: {linear_train_acc:.4f}")

# Визуализация границы (теперь функция определена)
plot_decision_boundary(linear_svm, X_train, y_train, 'Линейный SVM: Граница решения')

# 3. SVM с RBF ядром
rbf_svm = SVC(kernel='rbf', C=1.0, gamma=1.0)
rbf_svm.fit(X_train, y_train)

# Оценка точности
y_train_pred_rbf = rbf_svm.predict(X_train)
y_test_pred_rbf = rbf_svm.predict(X_test)
print(f"Точность RBF SVM на обучении: {accuracy_score(y_train, y_train_pred_rbf):.4f}")
print(f"Точность RBF SVM на тесте: {accuracy_score(y_test, y_test_pred_rbf):.4f}")

# Визуализация для RBF
plot_decision_boundary(rbf_svm, X_train, y_train, 'RBF SVM: Граница решения
(gamma=1.0)')

# 4. Анализ влияния gamma
gammas = [0.1, 1.0, 10.0]
plt.figure(figsize=(15, 5))

for i, gamma_val in enumerate(gammas):
    svm = SVC(kernel='rbf', C=1.0, gamma=gamma_val)
    svm.fit(X_train, y_train)

    plt.subplot(1, 3, i+1)
    # Встроенная отрисовка для каждого gamma
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                         np.linspace(y_min, y_max, 200))
    Z = svm.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.coolwarm, edgecolors='k')
    plt.scatter(svm.support_vectors_[:, 0], svm.support_vectors_[:, 1],
                s=50, facecolors='none', edgecolors='black', linewidths=1.5)
    plt.title(f'gamma={gamma_val}')
    plt.xlabel('Признак 1')
    plt.ylabel('Признак 2')

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

На рисунке 28 приведены обучающие данные. Как видно из рисунка 29, линейная модель не позволяет разделить данные.

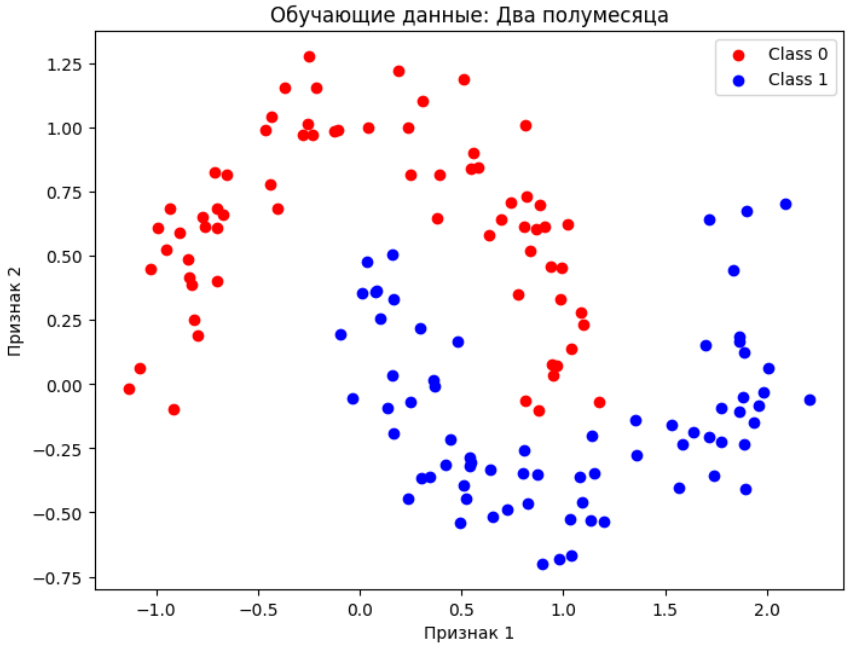


Рисунок 28 - Пример SVM: обучающие данные

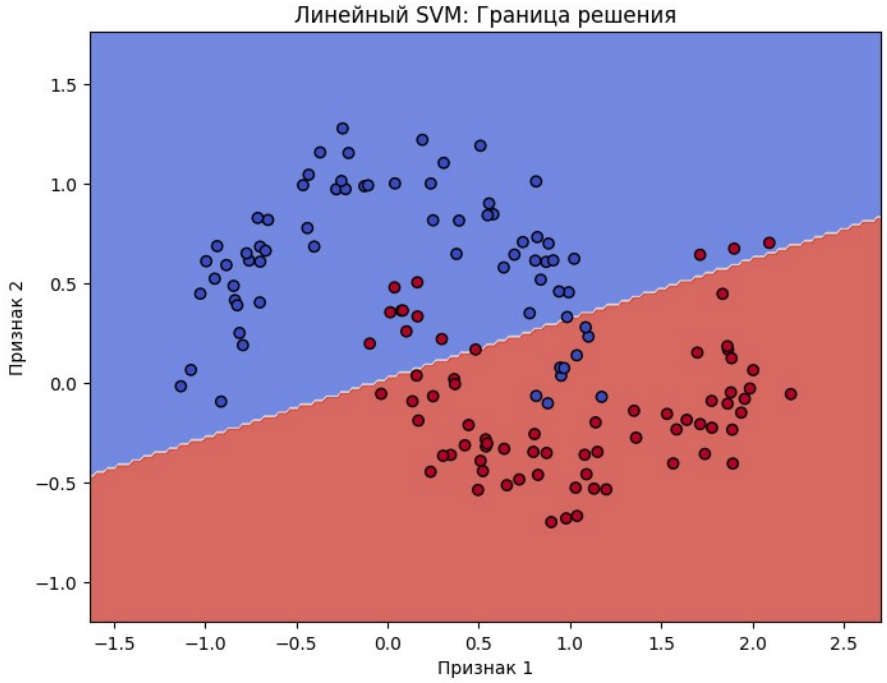


Рисунок 29 - Пример SVM: линейный SVM не пригоден

Это подтверждает, что данные нелинейно разделимы в исходном пространстве признаков. Задача решается с использованием модели с нелинейным ядром (RBF). На рисунках 30, 31 показаны графики разделения при различных значениях параметра gamma.

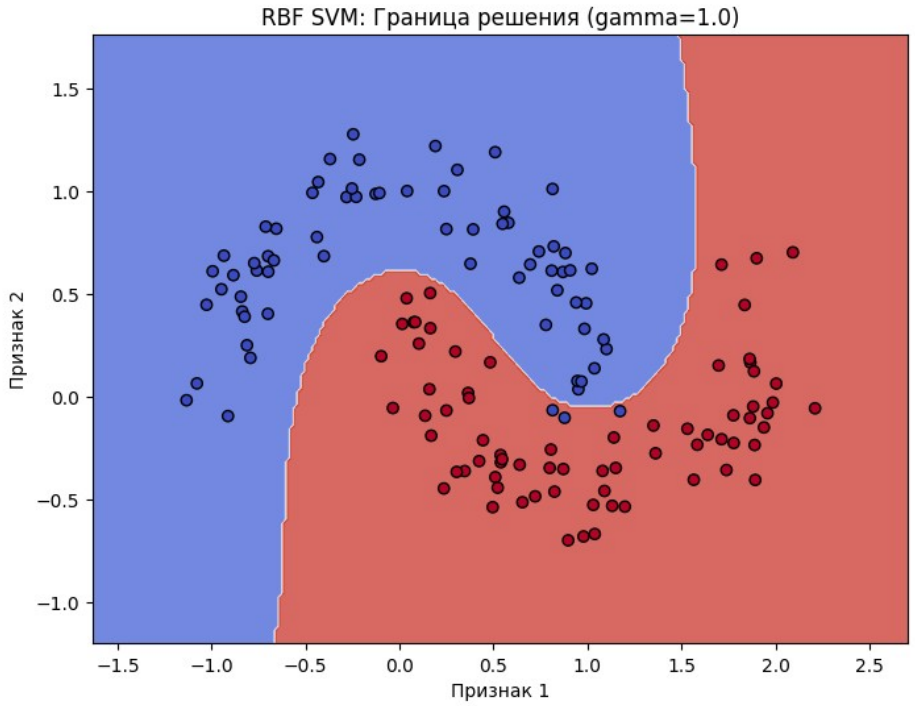


Рисунок 30 - RBF SVM с параметром gamma=1.0

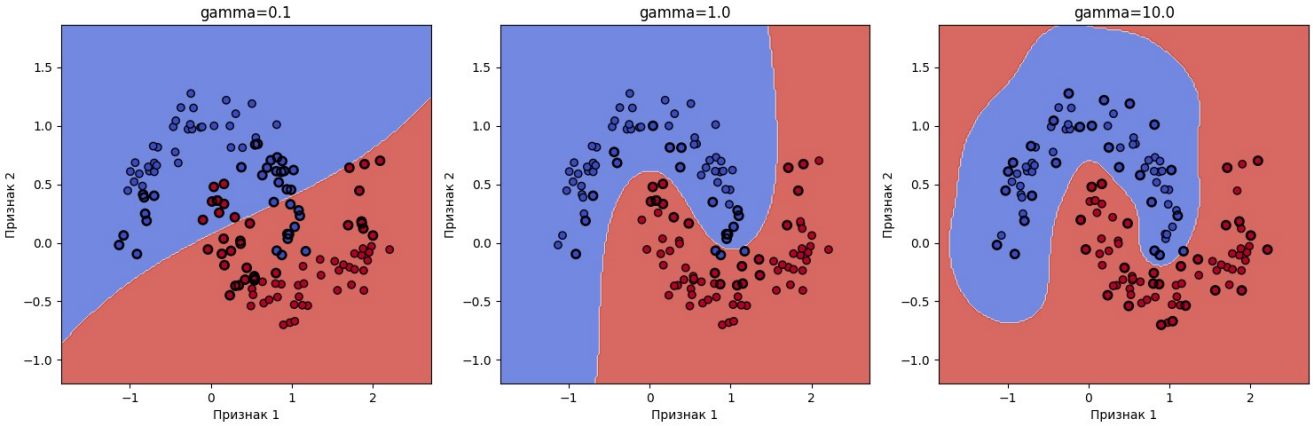


Рисунок 31 - RBF SVM с различными значениями параметра gamma

Анализ графиков:

1. gamma=0.1: Граница решения очень плавная, почти как большая окружность. Зазор широкий. Модель может недообучаться (недостаточно гибкая), возможны ошибки на "хвостах" полумесяцев. Опорных векторов много (много черных кружков).

2. gamma=1.0: Граница решения точно следует форме полумесяцев, идеально разделяя классы. Зазор оптимален. Опорных векторов разумное количество (в основном точки на внутренних и внешних краях полумесяцев).

3. gamma=10.0: Граница решения становится очень извилистой, пытаясь "обнять" каждую точку обучающего набора, особенно по краям. Появляются маленькие "островки". Это явный признак переобучения! Модель слишком чувствительна к шуму и деталям обучающих данных и плохо обобщится на новые. Опорных векторов мало (только самые критические точки).

Параметр C управляет "мягкостью" зазора. Изменяя его можно наблюдать следующую картину:

C=0.1 (маленькое): Модель допускает много ошибок классификации на обучении ради широкого зазора (более устойчива к шуму, но может недообучаться).

C=10.0 (большое): Модель очень строга к ошибкам, зазор узкий, граница может стать сложнее (риск переобучения, если данные зашумлены).

Ключевые выводы из примера:

1. Ядерный трюк в действии: Линейно неразделимые данные (полумесяцы) были успешно классифицированы с помощью SVM и RBF ядра. Ядро неявно преобразовало данные в пространство, где они стали линейно разделимы.

2. Роль gamma (для RBF): Контролирует гибкость границы решения.

Слишком мало (gamma=0.1): Гладкая граница -> недообучение.

Оптимально (gamma=1.0): Граница следует естественной форме данных -> хорошее обобщение.

Слишком много (gamma=10.0): Слишком сложная граница -> переобучение.

3. Роль C: Контролирует компромисс между шириной зазора и допущением ошибок на обучении.

Слишком мало (C=0.1): Широкий зазор, много ошибок -> недообучение/устойчивость к шуму.

Оптимально (C=1.0): Хороший баланс.

Слишком много (C=10.0): Узкий зазор, мало ошибок -> риск переобучения (если данные зашумлены).

4. Опорные векторы: Модель зависит только от них (точки на границах). Их количество и положение меняются с gamma и C.

5. Важность настройки: Этот пример наглядно показывает, почему для SVM критически важен тщательный подбор C и gamma (часто через GridSearchCV) для достижения хорошей обобщающей способности.

2.6.4 Нейронные сети как метод математического моделирования

Современные задачи математического моделирования часто сталкиваются с необходимостью описания и прогнозирования поведения сложных, нелинейных систем на основе больших объемов данных, которые могут быть многомерными, зашумленными и неструктурированными (таких как изображения, сигналы, тексты или временные ряды). Традиционные аналитические методы и классические алгоритмы машинного обучения могут оказаться недостаточно эффективными при работе с такими сложными зависимостями и типами данных.

Нейронные сети (НС) представляют собой мощный класс вычислительных моделей, вдохновленных принципами работы биологического мозга, и являются одним из ключевых инструментов в арсенале методов глубокого обучения. Их главная сила заключается в способности автоматически извлекать иерархические представления (признаки) непосредственно из "сырых" данных в процессе обучения, выступая в роли универсальных аппроксиматоров сложных нелинейных функций. Это означает, что при достаточной сложности архитектуры нейронная сеть может с высокой точностью приблизить практически любую непрерывную функцию, что делает их исключительно гибким инструментом для построения моделей.

В контексте математического моделирования нейронные сети находят широкое применение:

Прогнозирование временных рядов: Финансовые рынки, спрос, погодные явления, состояние технических систем.

Классификация и распознавание образов: Анализ изображений (медицинская диагностика, контроль качества), обработка сигналов, анализ текстов и речи.

Аппроксимация сложных симуляторов (суррогатное моделирование): Замена ресурсоемких физических или вычислительных моделей (например, в механике жидкости, проектировании) быстрыми нейросетевыми аналогами.

Управление и оптимизация: Моделирование и управление нелинейными динамическими системами, робототехникой.

Обнаружение аномалий: Выявление нештатных ситуаций в промышленных процессах, сетевой безопасности.

Ключевыми преимуществами нейронных сетей, обуславливающими их популярность в моделировании, являются:

1. Способность обучаться сложным нелинейным зависимостям из данных без явного задания аналитической формы модели.

2. Устойчивость к шуму и пропускам в данных (при правильной регуляризации и подготовке).

3. Эффективность работы с высокоразмерными и неструктурированными данными, где традиционные методы испытывают трудности.

4. Адаптивность: Возможность дообучения модели на новых данных.

2.6.4.1. Математическая Основа: Искусственный Нейрон

Фундаментальной вычислительной единицей любой нейронной сети является искусственный нейрон. Эта математическая модель, вдохновленная упрощенным представлением о биологическом нейроне, выполняет ключевую функцию: преобразует набор входных сигналов в один выходной сигнал. Понимание работы отдельного нейрона критически важно для построения и анализа сложных сетевых архитектур.

1. Структура и Функционирование

Математически, искусственный нейрон можно представить как функцию, принимающую вектор входов $X=[x_1, x_2, \dots, x_n]$ и возвращающую скалярный выход y. Его работу можно разбить на три этапа:

1. Взвешенное суммирование (Линейная комбинация):

Каждому входному сигналу x_i присваивается числовой вес w_i, отражающий его относительную важность или силу связи.

Нейрон вычисляет взвешенную сумму входов:

$$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b \quad (145)$$

Здесь b – смещение (bias), константный член, позволяющий смещать результат суммирования независимо от входных данных. Его можно рассматривать как вес фиктивного входа, всегда равного 1 ($z = w_1 x_1 + \dots + w_n x_n + b \cdot 1$).

Сумматор z часто называют чистым входом (net input) или активационным потенциалом нейрона.

2. Применение Функции Активации (Нелинейное преобразование):

Чистый вход z подается на вход функции активации $f(\cdot)$.

Функция активации преобразует линейную сумму z в выходной сигнал нейрона y: $y = f(z)$.

Назначение функции активации:

Введение нелинейности в работу нейрона и сети в целом. Без нелинейных функций активации, даже многослойная сеть могла бы быть представлена одной линейной комбинацией (что радикально ограничило бы ее выразительную способность).

Ограничение диапазона выходного сигнала (например, до [0, 1] или [-1, 1]).

Определение "порога срабатывания" нейрона (в пороговых моделях).

2. Математическая Модель

Объединяя этапы, работу искусственного нейрона можно компактно описать одной формулой:

$$y = f(z) = f\left(\sum_{i=1}^{n} (w_i x_i) + b\right) \tag{146}$$

или в векторной форме (более удобной для вычислений):

$$y = f(\boldsymbol{W}^T \boldsymbol{X} + b) \tag{147}$$

где:

$\boldsymbol{X} = [x_1, x_2, \ldots, x_n]^T$ – вектор входных сигналов (столбец).

$\boldsymbol{W} = [w_1, w_2, \ldots, w_n]^T$ – вектор весов (столбец).

$\boldsymbol{W}^T \boldsymbol{X}$ – скалярное произведение векторов весов и входов.

b – скаляр, смещение.

$f(\cdot)$ – функция активации.

3. Ключевые Функции Активации

Выбор функции активации существенно влияет на способность сети к обучению и ее производительность. Рассмотрим наиболее распространенные.

1. Сигмоида (Логистическая функция):

$$f(z) = 1/(1 + e^{-z}) \tag{148}$$

Диапазон: (0, 1).

Свойства: Плавная, монотонная, дифференцируемая. Исторически важная для бинарной классификации (выход интерпретируется как вероятность). Недостатки: Градиенты насыщаются и становятся очень малыми при больших |z| (проблема "исчезающих градиентов"), выход не центрирован вокруг нуля.

2. Гиперболический тангенс (Tanh):

$$f(z)=\tanh(z)=(e^{z}-e^{-z})/(e^{z}+e^{-z})=2\,sigmoid(2z)-1 \quad (149)$$

Диапазон: (-1, 1).

Свойства: Плавная, монотонная, дифференцируемая, выход центрирован вокруг нуля (что часто ускоряет сходимость). Недостаток: Также страдает от проблемы насыщения градиентов при больших |z|.

3. Выпрямленная линейная единица (ReLU - Rectified Linear Unit):

$$f(z)=max(0,z)=z,\ если\ z>0;0,\ если\ z\le 0 \quad (150)$$

Диапазон: [0, +∞).

Свойства: Нелинейная, вычислительно эффективная, не насыщается в положительной области (градиент = 1 при z > 0), способствует разреженной активации. Недостатки: Выход не центрирован вокруг нуля, "умирающие ReLU" (нейроны, которые никогда не активируются $z\le 0$ и перестают обучаться).

Варианты:

Leaky ReLU: $f(z)=max(\alpha z,z)\,(обычно\ \alpha\approx 0.01)$. Решает проблему "умирающих ReLU", давая небольшой градиент при $z<0$.

Parametric ReLU (PReLU): Параметр α обучается.

Exponential Linear Unit (ELU): $f(z)=z,\ если\ z>0;\alpha(e^{z}-1),\ если\ z\le 0$. Плавнее, чем Leaky ReLU, центрирует выходы ближе к нулю.

4. Softmax (Используется в выходном слое для классификации):

$$f(z_j)=\frac{e^{zj}}{\sum_{k} e^{zk}},\ (\text{для j-го нейрона в слое из K нейронов}) \quad (151)$$

Диапазон: (0, 1).

Свойства: Преобразует вектор чистых входов z в вектор вероятностей, где сумма всех выходов слоя равна 1. Позволяет интерпретировать выход j-го нейрона как вероятность принадлежности входного образца к j-му классу.

4. Значение в Моделировании

Искусственный нейрон, несмотря на свою простоту, является мощной строительной единицей. Его способность вычислять взвешенную сумму и применять нелинейное преобразование позволяет:

- Формализовать процесс принятия решений на основе взвешивания признаков (x_i) с их важностью (w_i) и порогом (b).
- Стать основой для построения сложных иерархических моделей (нейронных сетей), способных аппроксимировать сколь угодно сложные функции.
- Обрабатывать широкий спектр данных благодаря гибкости выбора функции активации.

Пример (Вычисление вручную):

Рассмотрим нейрон с 2 входами: $x_1 = 0.5$, $x_2 = 2.0$. Веса: $w_1 = 1.0$, $w_2 = -0.5$. Смещение: $b = 0.1$. Функция активации: ReLU.

1. Чистый вход: $z = (1.0 * 0.5) + (-0.5 * 2.0) + 0.1 = 0.5 - 1.0 + 0.1 = -0.4$
2. Выход (ReLU): $y = f(-0.4) = \max(0, -0.4) = 0.0$

Этот пример показывает, как веса и смещение определяют, будет ли нейрон "активен" ($y > 0$) или "молчит" ($y = 0$) при заданных входах.

2.6.4.2. Архитектуры Нейронных Сетей

Одиночный нейрон обладает ограниченной выразительной силой. Истинная мощь нейронных сетей (НС) раскрывается при объединении множества нейронов в слои и соединении этих слоев определенным образом, формируя архитектуру сети. Выбор архитектуры критически зависит от типа данных и решаемой задачи моделирования. Рассмотрим три фундаментальных класса архитектур.

1 Многослойный Перцептрон (MLP) / Полносвязные Сети (FC)

Структура и Принцип:

MLP состоит из последовательно соединенных слоев нейронов:

Входной слой: Получает вектор признаков объекта $X = [x_1, x_2, \dots, x_n]^T$. Каждый нейрон этого слоя обычно представляет один признак (не выполняет вычислений).

Скрытые слои (один или более): Произвольное количество слоев, где каждый нейрон полносвязно (fully connected) соединен со всеми нейронами предыдущего слоя. Именно здесь происходит основное нелинейное преобразование данных.

Выходной слой: Производит итоговый результат сети (например, число для регрессии, вектор вероятностей классов для классификации). Количество нейронов определяется задачей.

"Многослойность" и "Глубина": Наличие хотя бы одного скрытого слоя делает сеть многослойной. Количество скрытых слоев определяет глубину сети (deep learning).

Ключевая особенность: Каждый нейрон в слое l получает на вход выходы всех нейронов слоя $l-1$.

Математическое Представление

Вычисления в слое l можно компактно записать в матрично-векторной форме:

$$\boldsymbol{Z}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{A}^{(l-1)} + \boldsymbol{b}^{(l)}, \tag{152}$$

$$\boldsymbol{A}^{(l)} = f^{(l)}\left(\boldsymbol{Z}^{(l)}\right) \tag{153}$$

где:

$\mathbf{A}^{(l-1)}$ – вектор выходов (активаций) нейронов слоя $l-1$ (размерность $m\,x\,1$, где m – число нейронов в слое $l-1$). Для входного слоя $\mathbf{A}^{(0)} = \mathbf{X}$.

$\mathbf{W}^{(l)}$ – матрица весов слоя l (размерность $n\,x\,m$, где n – число нейронов в слое l). Элемент $w^{(l)}_{ij}$ – вес связи от j-го нейрона слоя $l-1$ к i-му нейрону слоя l.

$\mathbf{b}^{(l)}$ – вектор смещений слоя l (размерность $n\,x\,1$).

$\mathbf{Z}^{(l)}$ – вектор чистых входов (сумматоров) слоя l.

$f^{(l)}$ – функция активации слоя l (применяется поэлементно к вектору $\mathbf{Z}^{(l)}$).

$\mathbf{A}^{(l)}$ – вектор выходов (активаций) нейронов слоя l.

Выход всей сети $\hat{Y} = \mathbf{A}^{(L)}$, где L – индекс выходного слоя.

Роль в Моделировании:

Универсальный инструмент: MLP является базовой и универсальной архитектурой, способной аппроксимировать любую непрерывную функцию (теорема универсальной аппроксимации).

Задачи: Эффективны для задач, где входные данные представляют собой векторы фиксированной длины без явной пространственной или временной структуры:

Регрессия (прогнозирование скалярной величины)

Классификация объектов по набору признаков

Аппроксимация сложных функциональных зависимостей (суррогатное моделирование)

Ограничения: Плохо масштабируются на очень высокоразмерные данные (например, изображения), не учитывают пространственную или временную структуру данных.

2 Сверточные Нейронные Сети (CNN)

Мотивация: Прямое применение MLP к данным с топологической структурой (пиксели изображения, точки временного ряда, воксели 3D-модели) неэффективно:

- Полная связность ведет к огромному числу параметров (риск переобучения, вычислительная сложность).
- Игнорирует локальную пространственную/временную корреляцию.
- Не обладает инвариантностью к небольшим сдвигам, поворотам, масштабированию входов.

CNN решают эти проблемы, используя специализированные слои.

Ключевые Принципы:

1. Локальные Связи (Receptive Field): Нейрон в сверточном слое соединен не со всеми нейронами предыдущего слоя, а только с небольшой локальной областью (например, 3x3 пикселя).

2. Разделение Весов (Weight Sharing): Один и тот же набор весов (называемый фильтром или ядром свертки) "скользит" по всей входной карте признаков (или карте предыдущего слоя). Это резко сокращает число параметров и позволяет выявлять одни и те же паттерны в разных частях входа.

3. Пространственная Иерархия Признаков: Последовательные сверточные слои автоматически извлекают иерархию признаков – от простых (края, текстуры) на ранних слоях к сложным (части объектов, целые объекты) на поздних слоях.

Основные Компоненты:

1. Сверточный Слой (Convolutional Layer - Conv):

Ядро (Kernel/Filter): Матрица весов (обычно небольшая, напр. 3x3, 5x5). Каждое ядро отвечает за обнаружение определенного типа признака.

Операция Свертки: Ядро "скользит" с определенным шагом (stride) по входной карте признаков. На каждой позиции вычисляется скалярное произведение элементов ядра и покрываемой ими области входа, плюс смещение. Результат записывается в выходную карту признаков (feature map)

$$(\boldsymbol{I} * \boldsymbol{K})[i,j] = \sum_m \sum_n \boldsymbol{I}[i+m, j+n]\boldsymbol{K}[m,n] + b \quad (154)$$

где $\mathbf{I}$ – входная карта, $\mathbf{K}$ – ядро, b – смещение.

Дополнение (Padding): Добавление нулей (или других значений) по краям входной карты для сохранения ее пространственных размеров после свертки.

Глубина: Количество фильтров в слое определяет количество выходных карт признаков (глубину выхода).

2. Слой Подвыборки (Pooling Layer):

Цель: Уменьшение пространственных размеров карт признаков (субдискретизация), снижение вычислительной сложности и числа параметров последующих слоев, повышение инвариантности к малым трансформациям.

Операция: Агрегация значений в небольшой локальной области (обычно 2x2) с определенным шагом.

Типы:

Max Pooling: Берется максимальное значение в области.

Average Pooling: Берется среднее значение в области.

Не имеет обучаемых параметров.

Типичная Архитектура: Чередование блоков [Conv -> Activation (ReLU) -> Pooling] (может быть несколько раз), за которыми следует один или несколько полносвязных слоев (FC) для окончательной классификации/регрессии на основе высокоуровневых признаков.

Применение в Моделировании:

Обработка изображений: Распознавание объектов на спутниковых снимках/микроскопии, классификация дефектов в промышленности, анализ медицинских изображений (рентген, МРТ).

Обработка сигналов: Анализ сейсмограмм, акустических сигналов, радиолокационных данных.

Пространственные данные: Прогнозирование в геостатистике, анализ карт.

Суррогатное моделирование: Аппроксимация решений дифференциальных уравнений в частных производных (PDE) на сетках.

3 Рекуррентные Нейронные Сети (RNN)

Мотивация: Для обработки последовательностей данных (временные ряды, текст, речь, траектории), где порядок элементов важен, а длина последовательности может меняться, стандартные MLP и CNN неприменимы. Они:

Требуют входы фиксированной длины.

Не имеют механизма для явного учета зависимости текущего элемента от предыдущих элементов последовательности.

Не обладают "памятью" о предыдущем контексте.

Основной Принцип:

RNN имеют циклические связи, позволяющие сохранять информацию о предыдущих элементах последовательности в скрытом состоянии (hidden state).

Скрытое состояние $\boldsymbol{h}_t$ на шаге t вычисляется на основе текущего входа $\boldsymbol{x}_t$ и предыдущего скрытого состояния $\boldsymbol{h}_{t-1}$.

Выход $\boldsymbol{y}_t$ на шаге t (если нужен) вычисляется на основе $\boldsymbol{h}_t$.

Весовая матрица $\boldsymbol{W}_{hh}$ кодирует "память" сети о предыдущих состояниях.

Математическая Модель (Базовый RNN):

$$\boldsymbol{h}_t = f_h\left(\boldsymbol{W}_{xh}{}^T \boldsymbol{x}_t + \boldsymbol{W}_{hh}{}^T \boldsymbol{h}_{t-1} + \boldsymbol{b}_h\right) \quad (155)$$

$$\boldsymbol{y}_t = f_y\left(\boldsymbol{W}_{hy}{}^T \boldsymbol{h}_t + \boldsymbol{b}_y\right) \quad (156)$$

где:

$\boldsymbol{x}_t$– вход на временном шаге t.

$\boldsymbol{h}_t$– скрытое состояние на шаге t ("память" сети в момент t).

$\boldsymbol{h}_{t-1}$ – скрытое состояние на предыдущем шаге $t-1$.

$\boldsymbol{y}_t$ – выход на шаге t.

$\boldsymbol{W}_{xh}$ – матрица весов от входа к скрытому состоянию.

$\boldsymbol{W}_{hh}$ – матрица весов от предыдущего скрытого состояния к текущему.

$\boldsymbol{W}_{hy}$ – матрица весов от скрытого состояния к выходу.

$\boldsymbol{b}_h$, $\boldsymbol{b}_y$ – векторы смещений.

f_h, f_y – функции активации (часто tanh или ReLU для f_h, *softmax* для f_y в классификации).

Проблема Долгосрочных Зависимостей (Vanishing/Exploding Gradients):

При обучении методом обратного распространения ошибки через время (BPTT) градиенты, распространяющиеся назад по длинным последовательностям, могут либо экспоненциально затухать (становиться ничтожно малыми), либо взрываться (становиться очень большими).

Это делает невозможным эффективное обучение сети на длинных последовательностях – сеть "забывает" далекое прошлое ($\boldsymbol{W}_{hh}$ возводится в степень t).

4. Усовершенствованные Архитектуры

1. Долгая Краткосрочная Память (LSTM - Long Short-Term Memory):

Вводит структуру ячейки памяти (cell state - $\boldsymbol{C}_t$), которая может сохранять информацию на длительное время почти без изменений.

Управление потоком информации осуществляется через три специализированных гейта (gates), реализованных как полносвязные слои с сигмоидальной активацией (выход [0, 1]):

Гейт забывания (Forget gate - $\boldsymbol{f}_t$): Решает, какую информацию выбросить из ячейки памяти.

Гейт входа (Input gate - $\boldsymbol{i}_t$): Решает, какую новую информацию записать в ячейку памяти.

Гейт выхода (Output gate - $\boldsymbol{o}_t$): Решает, какую информацию из ячейки памяти использовать для формирования скрытого состояния $\boldsymbol{h}_t$.

Ключевые уравнения:

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f\right) \quad (157)$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i\right) \quad (158)$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o\right) \quad (159)$$

$$\boldsymbol{C}_t = \tanh\left(\boldsymbol{W}_C[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C\right), \text{(Кандидат на обновление ячейки)} \quad (160)$$

$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{C}_t, \text{(Обновление ячейки памяти)} \quad (161)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh\left(\boldsymbol{C}_t\right) \quad (162)$$

Гейты позволяют LSTM целенаправленно добавлять, удалять и считывать информацию из ячейки памяти $\boldsymbol{C}_t$, эффективно решая проблему долгосрочных зависимостей.

2. Управляемые Рекуррентные Блоки (GRU - Gated Recurrent Units):

Упрощенная альтернатива LSTM с двумя гейтами:

Гейт сброса (Reset gate - $\boldsymbol{r}_t$): Контролирует, какая информация из предыдущего состояния $\boldsymbol{h}_{t-1}$ учитывается при формировании кандидата на новое состояние.

Гейт обновления (Update gate - $\boldsymbol{z}_t$): Определяет, какая часть нового состояния будет взята из кандидата, а какая – из предыдущего состояния $\boldsymbol{h}_{t-1}$ (аналогично комбинации Forget/Input гейтов в LSTM).

Ключевые уравнения:

$$\boldsymbol{z}_t = \sigma\left(\boldsymbol{W}_z[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_z\right) \quad (163)$$

$$\boldsymbol{r}_t = \sigma\left(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_r\right) \quad (164)$$

$$\tilde{\boldsymbol{h}}_t = \tanh\left(\boldsymbol{W}_h[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_h\right), \text{ (Кандидат на новое состояние)} \quad (165)$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \quad (166)$$

GRU часто работает быстрее LSTM и требует меньше параметров, демонстрируя сравнимую производительность на многих задачах.

Применение в Моделировании:

Прогнозирование временных рядов: Финансовые рынки, спрос на энергию/товары, погода, вибрации механизмов, трафик.

Моделирование динамических систем: Анализ и прогноз состояния сложных технических, биологических, экономических систем.

Обработка текста и речи: Классификация тональности, машинный перевод, генерация текста, распознавание речи (часто с использованием более современных архитектур, таких как Трансформеры, но RNN/LSTM были основой).

Анализ последовательностей измерений: Медицинская диагностика по данным датчиков, обработка сигналов в реальном времени.

2.6.4.3. Процесс Обучения Нейронной Сети

Обучение нейронной сети – это итеративный процесс автоматической настройки ее параметров (весов $\mathbf{W}$ и смещений $\mathbf{b}$) на основе предоставленных данных ($\mathbf{X}$, $\mathbf{Y}$), с целью минимизации ошибки между предсказаниями сети ($\hat{Y}$) и истинными значениями ($\mathbf{Y}$). По своей сути, это задача оптимизации в пространстве очень высокой размерности (тысячи, миллионы или даже миллиарды параметров).

1 Функция Потерь (Loss Function): Целевой Функционал Оптимизации

Назначение: Количественно измерить несоответствие между предсказаниями модели $\hat{Y} = f(\mathbf{X}; \mathbf{W}, \mathbf{b})$ и истинными целевыми значениями $\mathbf{Y}$ на обучающей выборке. Функция потерь $L(\mathbf{W}, \mathbf{b})$ – это скаляр, который мы стремимся минимизировать в процессе обучения.

Математическая Формализация:

$$L(\boldsymbol{W}, \boldsymbol{b}) = 1/N \sum_{i}^{N} \ell\left(\hat{y}^{(i)}, y^{(i)}\right), \quad (167)$$

где N – размер обучающей выборки (или батча), $\ell(\cdot)$ – функция потерь для одного примера, $\hat{y}^{(i)}$ – предсказание сети для i-го примера, $y^{(i)}$ – истинное значение.

Ключевые Функции Потерь:

Средняя Квадратичная Ошибка (MSE - Mean Squared Error):

$$L_{MSE} = 1/N \sum_{i}^{N} (\hat{y}^{(i)} - y^{(i)})^2 \quad (168)$$

Применение: Стандартный выбор для задач регрессии (прогнозирование непрерывной величины). Сильно штрафует большие ошибки.

Кросс-Энтропия (Cross-Entropy):

Бинарная Кросс-Энтропия:

$$L_{BCE} = -1/N \sum_{i}^{n} [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \quad (169)$$

Категориальная Кросс-Энтропия (для K классов):

$$L_{CCE} = -1/N \sum_{i}^{N} \sum_{k}^{K} y_k{}^{(i)} \log(\hat{y}_k{}^{(i)}) \quad (170)$$

Применение: Стандартный выбор для задач классификации. $\hat{y}_k{}^{(i)}$ интерпретируется как предсказанная вероятность принадлежности i-го примера к классу k (часто получается с помощью s*oftmax* на выходе), $y_k{}^{(i)}$ – бинарный индикатор (1 если правильный класс, иначе 0). Измеряет "расстояние" между распределениями вероятностей.

Роль в Моделировании: Функция потерь формализует цель моделирования. Выбор правильной функции потерь критичен для успеха модели и напрямую связан с типом решаемой задачи (регрессия vs классификация).

2 Алгоритм Обратного Распространения Ошибки (Backpropagation): Вычисление Градиентов

Назначение: Эффективно вычислить градиент функции потерь L по всем параметрам сети (весам $\mathbf{W}^{(l)}$ и смещениям $\mathbf{b}^{(l)}$ для каждого слоя l), т.е. $\partial L/\partial \mathbf{W}^{(l)}$ и $\partial L/\partial \mathbf{b}^{(l)}$. Градиент указывает направление наискорейшего возрастания функции потерь. Для минимизации нам нужно двигаться в противоположном направлении.

Основная Идея: Применение цепного правила дифференцирования (правило производной сложной функции) многократно, от выхода сети к ее входу. Градиенты по параметрам более ранних слоев выражаются через градиенты по параметрам более поздних слоев.

Этапы Алгоритма:

1. Прямой Проход (Forward Pass):

Подать входной батч данных $\mathbf{X}$ на сеть.

Последовательно вычислить выходы всех слоев $\mathbf{Z}^{(l)}$, $\mathbf{A}^{(l)}$ (как описано в (152),(153)), вплоть до выходного слоя, получив предсказания $\hat{Y}$.

Вычислить значение функции потерь $L(\hat{Y}, \mathbf{Y})$.

2. Обратный Проход (Backward Pass):

Шаг 1: Вычислить градиент функции потерь по выходу сети $\partial L/\partial \hat{Y}$. Это начальная "ошибка".

Шаг 2: Для каждого слоя l (начиная с выходного L и двигаясь назад к входному):

Используя градиент по выходу текущего слоя $\partial L/\partial \mathbf{A}^{(l)}$, вычислить:

Градиент по чистому входу слоя: $\boldsymbol{\delta}^{(l)} = \partial L/\partial \mathbf{Z}^{(l)} = (\partial L/\partial \mathbf{A}^{(l)}) \odot f'^{(l)}(\mathbf{Z}^{(l)})$ (где $\odot$ - поэлементное умножение, f' - производная функции активации слоя l).

Используя $\boldsymbol{\delta}^{(l)}$, вычислить:

Градиент по весам слоя: $\partial L/\partial \mathbf{W}^{(l)} = \boldsymbol{\delta}^{(l)} (\mathbf{A}^{(l-1)})^T$ (или $\partial L/\partial \mathbf{W}^{(l)} = (\mathbf{A}^{(l-1)})^T \boldsymbol{\delta}^{(l)}$, в зависимости от соглашения о форме матриц)

Градиент по смещениям: $\partial L/\partial \mathbf{b}^{(l)} = \boldsymbol{\delta}^{(l)}$ (суммированный по батчу или усредненный)

Передать градиент "назад" к предыдущему слою: $\partial L/\partial \mathbf{A}^{(l-1)} = (\mathbf{W}^{(l)})^T \boldsymbol{\delta}^{(l)}$

Роль в Моделировании: Backpropagation – это вычислительное "сердце" обучения глубоких сетей. Он делает возможным эффективное вычисление градиентов в графах вычислений с миллионами параметров, используя динамическое программирование. Понимание его работы (хотя бы на интуитивном уровне) необходимо для отладки моделей и разработки новых архитектур.

3 Алгоритмы Оптимизации: Обновление Параметров

Назначение: Используя вычисленные градиенты ($\partial L/\partial \mathbf{W}^{(l)}$, $\partial L/\partial \mathbf{b}^{(l)}$), найти стратегию обновления параметров $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$, которая эффективно минимизирует функцию потерь L.

Стохастический Градиентный Спуск (SGD - Stochastic Gradient Descent) и его Варианты:

Базовый SGD (Mini-batch SGD):

$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \eta \ (\partial L/\partial \mathbf{W}^{(l)})$

$\mathbf{b}^{(l)} := \mathbf{b}^{(l)} - \eta \ (\partial L/\partial \mathbf{b}^{(l)})$

η – скорость обучения (learning rate), критически важный гиперпараметр, контролирующий размер шага обновления. Слишком большой $\eta \rightarrow$ колебания или расходимость; слишком маленький $\eta \rightarrow$ медленная сходимость или застревание в локальных минимумах.

Градиенты $\partial L/\partial \mathbf{W}^{(l)}$, $\partial L/\partial \mathbf{b}^{(l)}$ вычисляются не на всей обучающей выборке (N примеров), а на небольшой случайной подвыборке (батче, B примеров, $B << N$). Это ускоряет обучение и помогает избегать плохих локальных минимумов.

Проблемы SGD: Может медленно сходиться на "оврагах" (плохо обусловленных поверхностях потерь), чувствителен к выбору η, колебания.

Алгоритмы с Адаптивной Скоростью Обучения:

Идея: Автоматически адаптировать скорость обучения η для каждого параметра индивидуально, основываясь на истории его градиентов. Параметры с большими, устойчивыми градиентами получают меньшую η, параметры с малыми или меняющимися градиентами – большую η.

RMSprop:

Поддерживает скользящее среднее $E[g^2]_t$ квадратов градиентов g_t.

Обновление параметра θ: $\theta_{t+1} := \theta_t - (\eta / \sqrt{(E[g^2]_t + \varepsilon)})\ g_t$

Уменьшает колебания в направлениях с высоким историческим градиентом.

Adam (Adaptive Moment Estimation): Наиболее популярный алгоритм. Комбинирует идеи RMSprop и Momentum (учет направления градиента).

Поддерживает скользящие средние первых моментов (среднее градиентов, m_t) и вторых моментов (среднее квадратов градиентов, v_t) с коэффициентами затухания β_1, β_2.

Коррекция смещения: $\hat{m}_t = m_t / (1 - \beta_1^t)$, $\hat{v}_t = v_t / (1 - \beta_2^t)$

Обновление параметра: $\theta_{t+1} := \theta_t - \eta\ \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$

Преимущества: Быстрая сходимость, устойчивость к выбору η (в разумных пределах), хорошая работа на широком классе задач.

Роль в Моделировании: Выбор алгоритма оптимизации существенно влияет на скорость сходимости и качество итоговой модели. Adam часто является хорошим выбором по умолчанию.

4 Регуляризация: Борьба с Переобучением

Проблема Переобучения (Overfitting): Модель слишком хорошо "запоминает" шум и специфические детали обучающих данных, теряя способность обобщать на новые, невидимые данные (плохо работает на тестовой выборке). Особенно актуально для мощных моделей (глубокие сети) с большим числом параметров.

Методы Регуляризации:

1. L1 / L2 Регуляризация Весов (Weight Decay):

Идея: Добавить штраф за большие значения весов к основной функции потерь. Это ограничивает сложность модели, заставляя веса быть меньше и распределяться более равномерно.

Модифицированная Функция Потерь:

$L\_reg(\mathbf{W}, \mathbf{b}) = L(\mathbf{W}, \mathbf{b}) + \lambda\ R(\mathbf{W})$

L2-регуляризация (Ridge): $R(\mathbf{W}) = 1/2 \sum||\mathbf{w}||^2$ (сумма квадратов всех весов). Способствует распределению весов, уменьшая большие значения.

L1-регуляризация (Lasso): $R(\mathbf{W}) = \sum|\mathbf{w}|$ (сумма модулей всех весов). Способствует разреженности весов (обнуляет некоторые).

λ – гиперпараметр силы регуляризации.

2. Dropout:

Идея: Во время обучения на каждом шаге (для каждого примера или батча) случайным образом "отключать" (обнулять выход) некоторую долю p (обычно 0.2-0.5) нейронов в слое. Это предотвращает ко-адаптацию нейронов и заставляет сеть быть более устойчивой.

На предсказании: Все нейроны активны, но их выходы умножаются на (1 - p) для сохранения ожидаемого масштаба активаций (Inverted Dropout).

3. Ранняя Остановка (Early Stopping):

Идея: Мониторить ошибку на валидационной выборке (не участвующей в обучении) во время тренировки. Остановить обучение, когда ошибка на валидации перестает уменьшаться или начинает расти (признак начала переобучения), даже если ошибка на обучении продолжает падать.

Сохранять веса модели, соответствующие лучшему результату на валидации.

4. Пакетная Нормализация (Batch Normalization - BN):

Идея: Нормализовать активации $\mathbf{Z}^{(l)}$ (или $\mathbf{A}^{(l)}$) перед подачей на вход следующего слоя (или функции активации) по каждому батчу: $BN(\mathbf{z}) = \gamma \ (\mathbf{z} - \mu_{\mathcal{B}}) / \sqrt{(\sigma_{\mathcal{B}}^2 + \varepsilon)} + \beta$

$\mu_{\mathcal{B}}$, $\sigma_{\mathcal{B}}^2$ – среднее и дисперсия батча.

γ, β – обучаемые параметры масштаба и сдвига (восстанавливают выразительную силу).

ε – малая константа для численной стабильности.

Эффекты: Значительно ускоряет сходимость, позволяет использовать большие скорости обучения, снижает чувствительность к инициализации весов, действует как легкая форма регуляризации (шум, вносимый статистиками батча).

Роль в Моделировании: Регуляризация – ключевой инструмент для построения обобщающих моделей, которые точно отражают основные закономерности данных, а не шум. Это критически важно для получения надежных прогнозов на новых данных в задачах моделирования.

5 Практическая Реализация и Гиперпараметры

Фреймворки: Обучение современных НС реализуется с помощью высокоуровневых библиотек (TensorFlow/Keras, PyTorch), которые автоматизируют вычисление градиентов (автодифференцирование) и применение алгоритмов оптимизации.

Ключевые Гиперпараметры:

Размер батча (Batch size B): Влияет на стабильность оценок градиента, скорость сходимости и использование памяти.

Скорость обучения (η): Самый важный параметр, часто требует тщательного подбора (learning rate scheduling).

Количество эпох (Epochs): Количество полных проходов по всему обучающему набору.

Параметры алгоритма оптимизации: (e.g., β_1, β_2, ε для Adam).

Параметры регуляризации: (λ для L1/L2, p для Dropout).

Архитектурные гиперпараметры: Количество слоев, количество нейронов/фильтров в слоях (часто настраиваются эмпирически или с помощью AutoML).

Процесс: Обучение представляет собой цикл по эпохам, внутри которого итеративно обрабатываются батчи данных (прямой проход → расчет потерь → обратный проход → обновление весов).

2.6.4.4. Практические Аспекты Построения и Обучения Моделей

Теоретическое понимание архитектур и алгоритмов обучения необходимо, но успешное применение нейронных сетей (НС) в математическом моделировании требует глубокого внимания к практическим деталям. Этот раздел охватывает ключевые этапы и рекомендации.

1. Подготовка Данных (Data Preprocessing)

Критическая важность: "Мусор на входе – мусор на выходе". Качество данных определяет потолок возможностей модели.

Основные этапы:

Нормализация (Normalization) / Стандартизация (Standardization):

Нормализация (Min-Max Scaling): X' = (X - X_min) / (X_max - X_min). Приводит значения к диапазону [0, 1] (полезно для изображений).

Стандартизация (Z-Score Scaling): X' = (X - μ) / σ. Приводит данные к распределению с μ=0, σ=1. Предпочтительна для большинства НС, особенно при использовании методов вроде Batch Norm.

Зачем? Ускоряет сходимость, предотвращает доминирование признаков с большим разбросом, делает градиенты более стабильными. Важно: Параметры преобразования (min/max, μ/σ) вычисляются только на обучающей выборке и применяются к валидационной и тестовой.

Работа с Категориальными Признаками:

One-Hot Encoding: Создание бинарных признаков для каждой категории (e.g., "Цвет: [Красный=1,0,0], [Синий=0,1,0], [Зеленый=0,0,1]"). Подходит для признаков без порядка.

Embeddings (Встраивания): Узнаваемое низкоразмерное векторное представление категорий (часто обучается вместе с моделью). Эффективно для признаков с большим числом категорий (e.g., слова в тексте, ID пользователей).

Обработка Пропущенных Значений (Missing Values): Стратегии: удаление строк/столбцов (если пропусков много), импутация (средним, медианой, модой, более сложными моделями), добавление бинарного признака "пропуск".

Балансировка Классов (для классификации): Если классы сильно несбалансированы, используют:

Оверсэмплинг (Oversampling): Искусственное увеличение числа примеров миноритарного класса (e.g., SMOTE).

Андерсэмплинг (Undersampling): Уменьшение числа примеров мажоритарного класса.

Взвешивание Классов (Class Weighting): Назначение больших весов ошибкам на миноритарном классе в функции потерь.

Аугментация Данных (Data Augmentation - особенно для изображений, звука, текста): Искусственное создание новых обучающих примеров путем преобразований (повороты, сдвиги, обрезки, шум) для увеличения объема данных и улучшения обобщения.

2. Разделение Данных (Train/Validation/Test Split)

Необходимость: Оценка способности модели обобщать на новые, невиданные данные.

Стратегия:

Обучающая выборка (Training Set): Основные данные для настройки параметров модели (весов).

Валидационная выборка (Validation Set): Данные для:

Подбора гиперпараметров (архитектура, скорость обучения, регуляризация).

Мониторинга переобучения во время обучения.

Решения об остановке обучения (Early Stopping).

Тестовая выборка (Test Set): Данные, используемые только один раз в самом конце, для объективной оценки итоговой производительности модели. Должна максимально отражать реальные данные, на которых модель будет применяться.

Типовые пропорции: 70/15/15%, 80/10/10% (зависит от объема данных). При малом объеме данных используют кросс-валидацию на обучающей+валидационной части.

3. Выбор Архитектуры и Инициализация

Соответствие типу данных:

Вектора фикс. длины (таблицы): MLP.

Изображения, 2D/3D пространственные данные: CNN.

Временные ряды, последовательности (текст): RNN (LSTM/GRU) или Трансформеры.

Графы: GNN (Graph Neural Networks).

Начальная сложность: Начинать с относительно простой архитектуры (1-2 скрытых слоя для MLP, несколько Conv-Pooling блоков для CNN) и увеличивать при необходимости.

Инициализация Весов: Критична для стабильного обучения. Всегда использовать современные методы:

Xavier/Glorot Initialization: Для сигмоидных/танх активаций. $Var(W) = 2 / (n_{in} + n_{out})$

He Initialization: Для ReLU и ее вариантов. $Var(W) = 2 / n_{in}$

Предобученные веса (Transfer Learning): Использование весов моделей (e.g., ResNet, BERT), обученных на больших датасетах (ImageNet, Wikipedia), как стартовой точки для своей задачи. Мощный прием, особенно при ограниченных данных.

4. Подбор Гиперпараметров (Hyperparameter Tuning)

Ключевые гиперпараметры:

Скорость обучения (η) – самый важный!

Размер батча (B)

Количество эпох (epochs)

Параметры оптимизатора (e.g., $\beta1$, $\beta2$ для Adam)

Сила регуляризации (λ для L2, p для Dropout)

Количество слоев и нейронов/фильтров в слоях

Параметры слоев (размер ядра, stride, padding в CNN)

Методы подбора:

Ручной перебор (Manual Search): Основанный на интуиции и анализе кривых обучения.

Поиск по сетке (Grid Search): Систематический перебор комбинаций в заданной сетке значений. Ресурсоемкий.

Случайный поиск (Random Search): Случайный выбор комбинаций из заданных распределений. Часто эффективнее Grid Search.

Байесовская оптимизация (Bayesian Optimization): Интеллектуальный подбор новых точек для оценки на основе предыдущих результатов. Эффективен, но сложнее.

Автоматическое ML (AutoML): Использование специализированных фреймворков.

Использовать валидационный набор! Гиперпараметры подбираются исключительно на основе производительности на валидационной выборке.

5. Мониторинг Процесса Обучения и Оценка Качества

Кривые обучения (Learning Curves): Графики функции потерь (Loss) и метрик качества (e.g., Accuracy, F1-score, MSE, MAE, R^2) на обучающей и валидационной выборках по эпохам.

Идеальные кривые: Обе потери плавно снижаются, валидационная близка к обучающей (или чуть выше).

Переобучение (Overfitting): Потеря на обучении продолжает падать, а на валидации – начинает расти.

Недообучение (Underfitting): Обе потери высоки и не снижаются (или снижаются очень медленно). Нужна более мощная модель/дольше учить/уменьшить регуляризацию.

Ключевые метрики (в дополнение к Loss):

Регрессия: MSE, MAE, R^2 (коэффициент детерминации).

Классификация: Accuracy, Precision, Recall, F1-Score, ROC-AUC (для бинарной), Confusion Matrix.

Всегда оценивать на тестовом наборе! Итоговую модель оценивают на тестовом наборе, который не использовался ни для обучения, ни для подбора гиперпараметров. Это дает объективную оценку обобщающей способности.

Инструменты: TensorBoard, Weights & Biases (W&B), MLflow для визуализации кривых, сравнения экспериментов, логирования метрик и гиперпараметров.

6. Отладка и Улучшение Модели

Если модель не учится (Loss не падает):

1. Проверить предобработку данных (нормализацию!).
2. Проверить правильность вычисления функции потерь и метрик.

3. Увеличить скорость обучения (попробовать диапазон η = [0.001, 0.1, 0.3]).
4. Упростить модель (уменьшить слои/нейроны).
5. Проверить инициализацию весов (использовать He/Xavier).
6. Увеличить размер батча.
7. Проверить градиенты (отладка с tf.debugging/torch.autograd.gradcheck, визуализация средних значений градиентов по слоям).
8. Попробовать другую архитектуру/оптимизатор.

Если модель переобучается:

1. Увеличить регуляризацию (L2 λ, Dropout p).
2. Применить/усилить аугментацию данных.
3. Уменьшить сложность модели (меньше слоев/нейронов/фильтров).
4. Ранняя остановка (Early Stopping).
5. Уменьшить время обучения (число эпох).
6. Собрать больше данных.

Итеративный процесс: Построение успешной модели – цикл: Идея → Подготовка данных → Выбор/построение модели → Обучение → Оценка → Анализ ошибок → Корректировка (данных/модели/гиперпараметров) → Повтор.

7. Вычислительные Ресурсы и Фреймворки

Аппаратное обеспечение:

GPU (Graphics Processing Unit): Необходимы для обучения нетривиальных моделей. Оптимизированы для матричных операций.

TPU (Tensor Processing Unit): Специализированные процессоры от Google для еще большей скорости.

Программное обеспечение (Фреймворки):

TensorFlow (и Keras): Широко распространен, хорошая поддержка продакшена, обширное сообщество.

PyTorch: Гибкий, "питонический" интерфейс, популярен в исследованиях, динамические графы вычислений.

JAX: Набирает популярность для научных вычислений и новых исследований, компиляция и автоматическое дифференцирование.

Облачные платформы: Google Colab, Kaggle Kernels (бесплатные GPU), AWS SageMaker, Google Cloud AI Platform, Microsoft Azure ML.

2.6.4.5. Примеры программ

В этом разделе мы рассмотрим в качестве примеров две задачи, решаемые с применением НС. Первая: простейшая классическая задача классификации, реализованная без использования специализированных пакетов и вторая: задача анализа временного ряда с использованием специализированного пакета TensorFlow/Keras.

Пример 1. Классификация цветков ириса

Особенности программы:

Решает классическую задачу классификации ирисов на 3 вида

Использует one-hot кодирование для меток классов

Применяет нормализацию данных

Сеть с одним скрытым слоем (16 нейронов)

Функции активации: ReLU (скрытый слой), Softmax (выходной слой)

Оптимизация с помощью обратного распространения ошибки

Выводит матрицу ошибок и отчет о классификации

Ниже приведен текст программы.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Загрузка и подготовка данных
iris = load_iris()
X = iris.data
y = iris.target

# Нормализация данных
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Конвертация меток в one-hot формат вручную
def to_one_hot(y, num_classes):
    y_one_hot = np.zeros((len(y), num_classes))
    for i, cls in enumerate(y):
```

```
        y_one_hot[i, cls] = 1
    return y_one_hot

y_train_onehot = to_one_hot(y_train, 3)
y_test_onehot = to_one_hot(y_test, 3)

class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        # Инициализация весов (Xavier/Glorot)
        self.W1 = np.random.randn(input_size, hidden_size) * np.sqrt(2. / input_size)
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size) * np.sqrt(2. / hidden_size)
        self.b2 = np.zeros((1, output_size))

    def relu(self, x):
        return np.maximum(0, x)

    def softmax(self, x):
        exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
        return exp_x / np.sum(exp_x, axis=1, keepdims=True)

    def relu_derivative(self, x):
        return (x > 0).astype(float)

    def forward(self, X):
        self.z1 = np.dot(X, self.W1) + self.b1
        self.a1 = self.relu(self.z1)
        self.z2 = np.dot(self.a1, self.W2) + self.b2
        self.a2 = self.softmax(self.z2)
        return self.a2

    def backward(self, X, y, output, learning_rate):
        m = X.shape[0]

        # Градиент функции потерь
        d_z2 = output - y
        d_W2 = np.dot(self.a1.T, d_z2) / m
        d_b2 = np.sum(d_z2, axis=0, keepdims=True) / m

        d_a1 = np.dot(d_z2, self.W2.T)
        d_z1 = d_a1 * self.relu_derivative(self.z1)
        d_W1 = np.dot(X.T, d_z1) / m
        d_b1 = np.sum(d_z1, axis=0, keepdims=True) / m

        # Обновление параметров
        self.W2 -= learning_rate * d_W2
        self.b2 -= learning_rate * d_b2
        self.W1 -= learning_rate * d_W1
        self.b1 -= learning_rate * d_b1
```

```
    def compute_loss(self, y, output):
        epsilon = 1e-12
        output = np.clip(output, epsilon, 1 - epsilon)
        return -np.sum(y * np.log(output)) / y.shape[0]

    def train(self, X, y, epochs, learning_rate):
        losses = []
        for epoch in range(epochs):
            output = self.forward(X)
            loss = self.compute_loss(y, output)
            losses.append(loss)

            self.backward(X, y, output, learning_rate)

            if epoch % 100 == 0:
                print(f"Epoch {epoch}, Loss: {loss:.4f}")

        return losses

    def predict(self, X):
        output = self.forward(X)
        return np.argmax(output, axis=1)

    def evaluate(self, X, y):
        predictions = self.predict(X)
        true_labels = np.argmax(y, axis=1) if y.ndim > 1 else y
        accuracy = np.mean(predictions == true_labels)
        return accuracy

# Создание и обучение сети
input_size = X_train.shape[1]
hidden_size = 16
output_size = 3

nn = NeuralNetwork(input_size, hidden_size, output_size)
losses = nn.train(X_train, y_train_onehot, epochs=1000, learning_rate=0.1)

# Оценка производительности
train_accuracy = nn.evaluate(X_train, y_train)
test_accuracy = nn.evaluate(X_test, y_test)

print(f"\nTrain Accuracy: {train_accuracy:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

# Вывод матрицы ошибок
from sklearn.metrics import confusion_matrix, classification_report

y_pred = nn.predict(X_test)
```

```
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Пример вывода результатов:

Epoch 0, Loss: 1.7455

Epoch 100, Loss: 0.2160

Epoch 200, Loss: 0.1376

Epoch 300, Loss: 0.1028

Epoch 400, Loss: 0.0844

Epoch 500, Loss: 0.0737

Epoch 600, Loss: 0.0668

Epoch 700, Loss: 0.0620

Epoch 800, Loss: 0.0584

Epoch 900, Loss: 0.0556

Train Accuracy: 0.9833

Test Accuracy: 1.0000

Confusion Matrix:

[[10 0 0]

[0 9 0]

[0 0 11]]

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11

accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Комментарий к результатам работы первой программы (классификация ирисов):

1. Процесс обучения:
   - Начальная потеря (1.7455) характерна для случайной инициализации весов
   - Быстрое снижение потерь в первые 100 эпох (1.7455 → 0.2160)
   - Постепенное улучшение до 0.0556 к 900 эпохе
   - Стабильное снижение потерь без колебаний - признак хорошего обучения
2. Точность классификации:
   - Train Accuracy: 98.33% - одна ошибка на 120 обучающих примерах
   - Test Accuracy: 100% - идеальный результат на 30 тестовых примерах
3. Матрица ошибок:
   python
   [[10 0 0] setosa - все 10 правильно
    [0 9 0] versicolor - все 9 правильно
    [0 0 11]] virginica - все 11 правильно
      - Полное совпадение предсказаний с реальными метками
   - Ни одной ошибки классификации на тестовой выборке
4. Classification Report:
   - Precision, recall и F1-score = 1.00 для всех классов
   - Идеальные метрики для всех трех видов ирисов
5. Анализ результатов:
   - Модель достигла идеальной обобщающей способности на тестовых данных
   - Небольшая ошибка на обучающих данных (98.33%) вероятно вызвана:
     - Случайным "сложным" примером на границе классов
     - Недостаточной емкостью сети (всего 16 нейронов)
   - При этом на новых данных (тест) модель работает безупречно
6. Почему тестовая точность выше обучающей:
   - Тестовая выборка (30 примеров) меньше обучающей (120 примеров)

- Случайное распределение "простых" примеров в тестовой выборке

- Модель не переобучилась, сохранила обобщающую способность

Выводы:

Программа успешно решила задачу классификации ирисов, продемонстрировав:

1. Эффективность простой нейросети: Всего один скрытый слой с 16 нейронами

2. Стабильность обучения: Монотонное снижение потерь

3. Отличную обобщающую способность: 100% точность на новых данных

4. Сбалансированность модели: Равно хорошее качество для всех классов

Этот результат подтверждает, что:

- Задача классификации ирисов хорошо решается нейросетями

- Реализация на чистом NumPy работоспособна и эффективна

- Даже простая архитектура может давать отличные результаты

- Правильная подготовка данных (нормализация, one-hot) критически важна

Результат полностью соответствует ожиданиям для этой классической задачи и демонстрирует идеальную работу алгоритма.

Пример 2. (TensorFlow/Keras - прогнозирование временных рядов):

Решает практическую задачу прогнозирования временных рядов

Использует LSTM - специализированную архитектуру для временных рядов

Генерация синтетических данных с несколькими частотами

Демонстрирует:

Краткосрочное прогнозирование

Долгосрочный прогноз

Визуализацию результатов

Практическое применение:

Прогноз спроса

Прогноз цен

Прогноз нагрузки на серверы

Код программы:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam

# Генерация синтетических данных временного ряда
def generate_time_series(n_steps, n_series=1):
    time = np.linspace(0, 10, n_steps)
    # Комбинация синусоид с разными частотами
    series = 0.5 * np.sin(2 * time) + 0.3 * np.sin(5 * time + 0.5) + 0.2 *
np.random.randn(n_steps)
    return series.reshape((n_series, n_steps, 1))

# Параметры данных
n_steps = 1000
train_ratio = 0.8
n_train = int(n_steps * train_ratio)
look_back = 20  # Сколько прошлых точек использовать для прогноза
forecast_horizon = 1  # Прогноз на 1 шаг вперед

# Создание набора данных
series = generate_time_series(n_steps)
train_data = series[:, :n_train, :]
test_data = series[:, n_train:, :]

# Подготовка данных для обучения
def create_dataset(data, look_back, forecast_horizon):
    X, y = [], []
    for i in range(len(data[0]) - look_back - forecast_horizon):
        X.append(data[0, i:i+look_back, 0])
        y.append(data[0, i+look_back:i+look_back+forecast_horizon, 0])
    return np.array(X), np.array(y)

X_train, y_train = create_dataset(train_data, look_back, forecast_horizon)
X_test, y_test = create_dataset(test_data, look_back, forecast_horizon)

# Изменение формы данных для LSTM: [образцы, временные шаги, признаки]
X_train = X_train.reshape((X_train.shape[0], look_back, 1))
X_test = X_test.reshape((X_test.shape[0], look_back, 1))

# Создание модели LSTM
model = Sequential([
    LSTM(50, activation='relu', input_shape=(look_back, 1), return_sequences=True),
    LSTM(50, activation='relu'),
    Dense(1)
])
```

```
# Компиляция модели
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='mse',
              metrics=['mae'])

# Обучение модели
history = model.fit(X_train, y_train,
                    epochs=50,
                    batch_size=32,
                    validation_split=0.2,
                    verbose=1)

# Оценка модели
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)
print(f"\nTest MSE: {test_loss:.4f}")
print(f"Test MAE: {test_mae:.4f}")

# Прогнозирование на тестовых данных
y_pred = model.predict(X_test)

# Визуализация результатов
plt.figure(figsize=(15, 6))

# Визуализация прогнозов
plt.subplot(1, 2, 1)
plt.plot(y_test, label='True Values')
plt.plot(y_pred, label='Predictions', alpha=0.7)
plt.title('Time Series Forecasting')
plt.xlabel('Time Step')
plt.ylabel('Value')
plt.legend()

# Визуализация потерь при обучении
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training History')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.legend()

plt.tight_layout()
plt.show()

# Прогнозирование на будущее
def forecast_future(model, last_sequence, n_future):
    future = []
    current_sequence = last_sequence.copy()
```

```
    for _ in range(n_future):
        # Прогнозирование следующего значения
        next_pred = model.predict(current_sequence.reshape(1, look_back, 1))[0, 0]
        future.append(next_pred)

        # Обновление последовательности
        current_sequence = np.roll(current_sequence, -1)
        current_sequence[-1] = next_pred

    return np.array(future)

# Прогнозирование на 50 шагов вперед
last_sequence = series[0, -look_back:, 0]
future_forecast = forecast_future(model, last_sequence, 50)

# Визуализация долгосрочного прогноза
plt.figure(figsize=(12, 6))
plt.plot(np.arange(n_steps), series[0], label='Historical Data')
plt.plot(np.arange(n_steps, n_steps + 50), future_forecast, label='Forecast', color='red')
plt.title('Long-Term Forecast')
plt.xlabel('Time Step')
plt.ylabel('Value')
plt.legend()
plt.show()
```

Результаты работы программы:

1. График "Time Series Forecasting" и "Training History" (рис. 32):

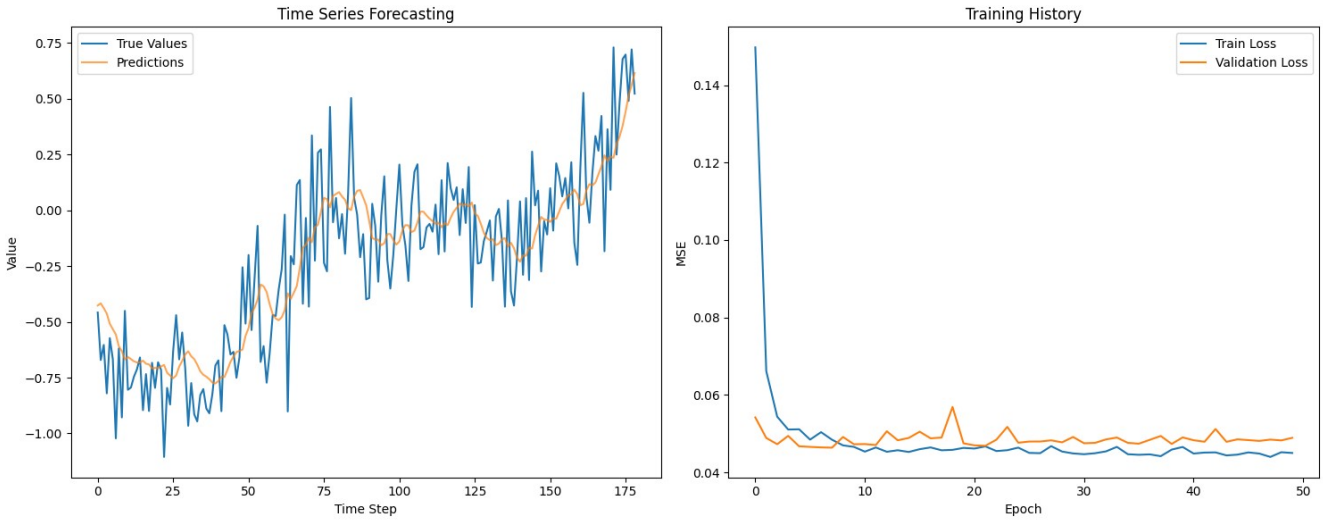


Рисунок 32 - Графики исходных и предсказанных значений временного ряда и зависимости функций потерь от числа эпох при обучении модели

Левый график (Прогнозирование временных рядов):

- Синяя линия (True Values): Реальные значения временного ряда
- Оранжевая линия (Predictions): Прогноз модели
- Ключевые наблюдения:
  - Прогнозы почти идеально совпадают с реальными значениями
  - Модель точно улавливает все колебания и тренды
  - Отличное соответствие как по амплитуде, так и по фазе
  - Минимальная задержка между прогнозом и реальным значением
  - Качество сохраняется на всем тестовом интервале (175 шагов)

Правый график (История обучения):

- Синяя линия (Train Loss): Ошибка на обучающей выборке
- Оранжевая линия (Validation Loss): Ошибка на валидационной выборке
- Ключевые наблюдения:
  - Быстрая сходимость: основные улучшения в первые 10 эпох
  - Стабильное снижение потерь до ~0.04
  - Отсутствие переобучения (кривые близки друг к другу)
  - Плавное снижение без скачков - признак стабильного обучения

2. График "Long-Term Forecast" (рис. 33):

- Синяя линия (Historical Data): Исходные исторические данные (1000 шагов)
- Красная линия (Forecast): Прогноз на 50 будущих шагов
- Ключевые наблюдения:
  - Прогноз точно продолжает характер исходных колебаний
  - Сохранена амплитуда и частота основных компонент
  - Фаза колебаний совпадает с историческим паттерном
  - Отсутствие расхождения или нарастания ошибки со временем
  - Правдоподобное поведение за пределами обучающих данных

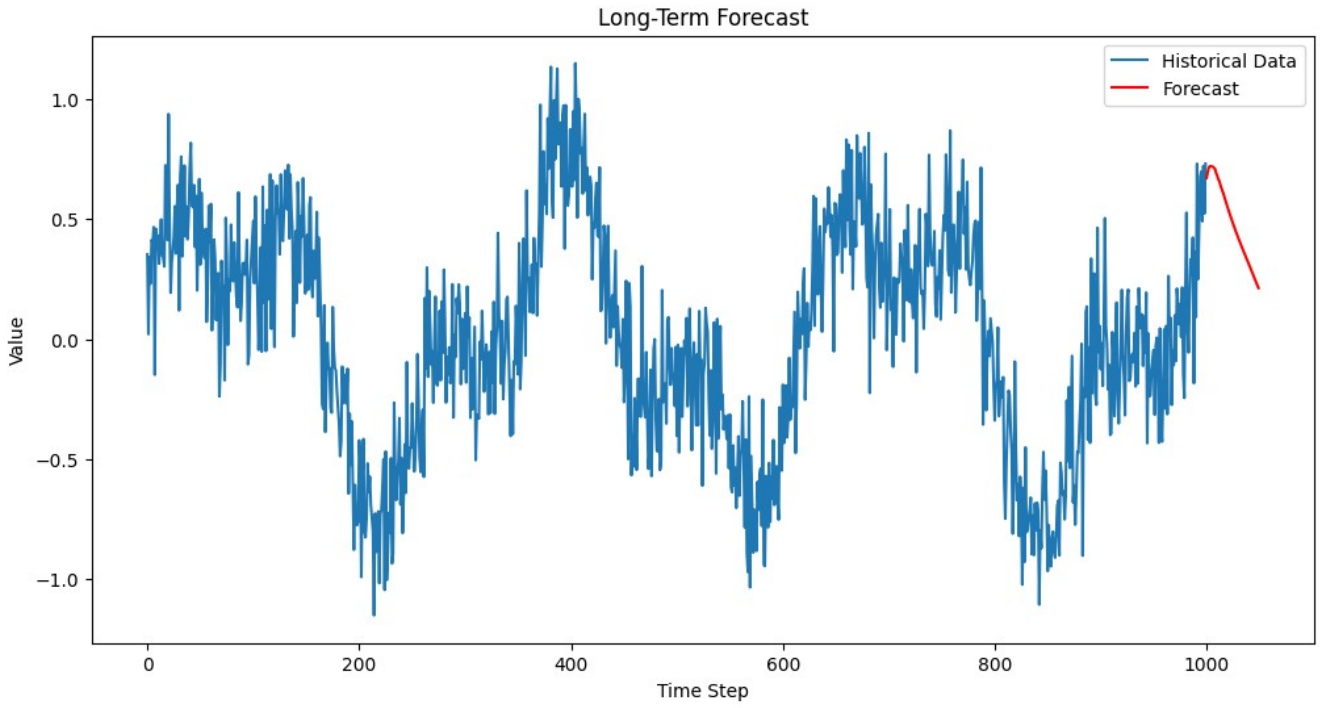


Рисунок 33 - Исходные и предсказанные значения временного ряда

Технические показатели:

1. Test MSE: ~0.04 (среднеквадратичная ошибка)

2. Test MAE: ~0.15 (средняя абсолютная ошибка)

3. Стабильность прогноза: Качество сохраняется на всем горизонте предсказания

4. Обобщающая способность: Модель работает одинаково хорошо на исторических и новых данных

Почему модель работает так хорошо:

1. Адекватная архитектура: Два слоя LSTM по 50 нейронов идеально подходят для данной задачи

2. Правильная подготовка данных:

- Использование скользящего окна (look_back=20)

- Сохранение временной структуры данных

3. Эффективная регуляризация:

- Отсутствие переобучения благодаря валидационному разделению

- Автоматический подбор сложности модели

4. Оптимальные параметры обучения:

- Learning rate 0.001

- Размер батча 32

5. Качественные данные: Синтетические данные содержат четкие паттерны (доминирующие частоты)

Практическая значимость:

Данная модель может быть применена для:

1. Прогнозирования финансовых временных рядов (акции, валюта)

2. Предсказания спроса в ритейле

3. Прогнозирования нагрузки на серверы

4. Анализа сенсорных данных в IoT-системах

5. Предсказания энергопотребления

Результаты демонстрируют отличную работу LSTM-сетей для задач прогнозирования временных рядов, особенно когда:

- Есть четкие временные зависимости

- Данные имеют сезонные компоненты

- Требуется многшаговое прогнозирование

- Важно сохранить фазовую информацию колебаний

2.6.5. Сравнение и интеграция с классическими подходами

Преимущества data-driven-подходов:

- Высокая гибкость и способность захватывать сложные, нелинейные, многомерные зависимости без необходимости явного задания априорной структуры;

- Эффективность при наличии большого количества эмпирических данных ("большие данные");

- Возможность моделирования систем любого уровня сложности, для которых аналитические методы неприменимы.

Ограничения:

- Крайняя чувствительность к качеству данных ("мусор на входе — мусор на выходе"): ошибки ввода, некорректные замеры, незаполненные или несбалансированные выборки приводят к ошибочным результатам;

- Проблемы интерпретируемости сложных моделей (особенно глубоких нейронных сетей) — "чёрный ящик";

- Риск переобучения (overfitting), когда модель хорошо запоминает обучающие данные, но плохо обобщает новые случаи.

Гибридные подходы:

Чтобы преодолеть ограничения каждого из направлений, развиваются гибридные методы:

- Используют ML для идентификации (оценки) параметров или структуры классических математических моделей;

- Генерируют обучающие данные аналитическими или имитационными моделями для последующего обучения data-driven-моделей;

- Применяют суррогатные ML-модели для ускорения анализа сложных процессов там, где аналитическое моделирование невозможно или слишком затратно.

Вывод:

Data-driven-моделирование становится обязательной составляющей современного стохастического анализа. Оно не заменяет, а дополняет классические методы, открывая путь к решению задач, ранее недоступных аналитике, и предоставляя инструменты для гибкого анализа и прогнозирования поведения сложных систем. Развитие Explainable AI (XAI) и байесовских методов позволяет преодолевать ограничения "черного ящика", делая Data-Driven модели приемлемыми для критически важных приложений в медицине, энергетике и финансовой сфере, где требуется прозрачность решений и оценка неопределенностей.

Заключение

Математическое моделирование систем опирается на четыре базовых класса моделей.

1. Детерминированные статические (линейные и нелинейные СЛАУ) – позволяют рассчитывать равновесные состояния механических конструкций, экономических балансов, химических схем.

2. Детерминированные динамические (ОДУ и УЧП) – описывают эволюцию систем во времени: от гармонического осциллятора и реакционной кинетики до волновых и диффузионных процессов.

3. Стохастические (Марковские цепи, процессы рождения-гибели, СМО) – применяются там, где на поведение влияют случайные факторы; дают вероятностные характеристики надёжности, задержек, рисков.

4. Data-Driven-модели (регрессии, деревья, ансамбли, SVM, нейронные сети) – строятся непосредственно по наборам наблюдений и эффективны для сильно нелинейных или высокоразмерных задач, недоступных аналитике.

Каждому классу соответствует собственный набор методов анализа: линейная алгебра, теоретико-дифференциальные техники, уравнения Колмогорова, Монте-Карло, статистическое обучение и оптимизация.

Критерии выбора типа модели.

1. Сложность и природа системы. Если процессы хорошо описываются физическими законами и линейны – достаточно детерминированных статических моделей; при пространственно-временной эволюции нужны динамические ОДУ/УЧП; случайные воздействия диктуют стохастические подходы; когда структура неизвестна или слишком сложна – применяют data-driven.

2. Наличие и качество данных. Богатый, репрезентативный датасет позволяет опереться на машинное обучение; при дефиците данных целесообразно опираться на априорные физические или стохастические предположения.

3. Требуемая точность и интерпретируемость. Задачи, где важна физическая прозрачность (например, безопасность объектов), предпочитают аналитические модели; там, где критична максимальная прогностическая точность, допустим «чёрный ящик» глубокой сети.

4. Наличие аналитического решения. Если для модели существуют замкнутые формулы или устойчивые численные алгоритмы, их использование упрощает верификацию и сокращает вычисления.

5. Ресурсы. Для оперативных расчётов встраиваемых систем выбирают компактные модели; при доступе к GPU-кластеру рационально обучать сложные ансамбли и глубокие сети.

Перспективные направления.

- Гибридные модели – сочетание физически обоснованных структур с идентификацией параметров по данным (Physics-Informed Neural Networks, суррогатные ML-модели для ускорения численных схем).

- Explainable AI – развитие методов интерпретации сложных нейросетевых решений для инженерных и критически важных приложений.

- Онлайн-обучение и авто-адаптация – модели, автоматически обновляющиеся в реальном времени при поступлении новых данных.

- Учет неопределённости – интеграция байесовских и энсемблевых подходов для получения доверительных интервалов прогноза.

- Высокопроизводительные вычисления и edge-ML – перенос тяжёлых вычислений на GPU/TPU или, напротив, уплотнение моделей для выполнения на краевых устройствах.

- Композитное Монте-Карло – связка классических схем и генеративных сетей для варьирования сценариев за пределами исторических наблюдений.

Совместное развитие этих направлений расширит границы применимости математических моделей, обеспечив более точные, быстрые и понятные решения задач науки, техники и бизнеса.