Курсовая работа

> Выполнил: Литти Тимофей

[Часть 1. Идеальный солнечный элемент](#)
[Часть 2. Реальная модель](#)
[Исправления](#)

Часть 1. Идеальный солнечный элемент

Вывод напряжения холостого хода (V_{oc})

Уравнение для идеальной ячейки имеет вид:

$$J = J_{sc} - J_0 \left(\exp\left(\frac{eV}{mk_BT} \right) - 1 \right)$$

Где:

- J (Плотность тока) - результирующий ток, протекающий через солнечный элемент, нормированный на единицу площади
- V (Напряжение) - Напряжение на клеммах солнечного элемента.
- J_{sc} (Плотность тока короткого замыкания) - Максимальный ток, который может выдать элемент, когда напряжение $V = 0$ (клеммы замкнуты накоротко).
- J_0 (Ток насыщения диода) - Ток утечки диода при обратном смещении в темноте.
- e (Элементарный заряд) - Заряд электрона. Константа, равная $1.6 \cdot 10^{-19}$ C (Кулон).
- k_B (Постоянная Больцмана) - Константа, связывающая температуру с энергией. Равна $1.38 \cdot 10^{-23}$ $\mathrm{J/K}$.
- T (Температура) - Абсолютная температура p-n перехода солнечного элемента в Кельвинах (K).
- m (Фактор идеальности диода) - Безразмерный коэффициент, который показывает, насколько реальный диод близок к идеальному уравнению Шокли. ($m = 1$: Идеальный диод (преобладает диффузионный ток); $m = 2$: Преобладает рекомбинационный ток в области обеднения.)

Напряжение холостого хода (V_{oc}) достигается, когда ток через ячейку равен нулю ($J = 0$). Подставим это в уравнение:

$$0 = J_{sc} - J_0 \left(\exp\left(\frac{eV_{oc}}{mk_BT} \right) - 1 \right)$$

Преобразуем уравнение для нахождения V_{oc}:

1. $\frac{J_{sc}}{J_0} = \exp\left(\frac{eV_{oc}}{mk_BT} \right) - 1$

2. $\exp\left(\frac{eV_{oc}}{mk_BT}\right) = \frac{J_{sc}}{J_0} + 1$
3. Так как $J_{sc} \gg J_0$, можно считать, что $\frac{J_{sc}}{J_0} + 1 \approx \frac{J_{sc}}{J_0}$.
4. Логарифмируем обе части: $\frac{eV_{oc}}{mk_BT} = \ln\left(\frac{J_{sc}}{J_0}\right)$

Итоговая формула, для определения V_{oc} как функции от m и J_{sc}.:

$$V_{oc} \approx \frac{mk_BT}{e} \ln\left(\frac{J_{sc}}{J_0}\right)$$

V_{oc} линейно зависит от фактора идеальности диода m и логарифмически от тока короткого замыкания J_{sc}.

Влияние m на максимальную мощность

Увеличение фактора идеальности m (от 1.5 до 2.5) приводит к следующим эффектам:

Как видно из формулы, V_{oc} растет пропорционально m.

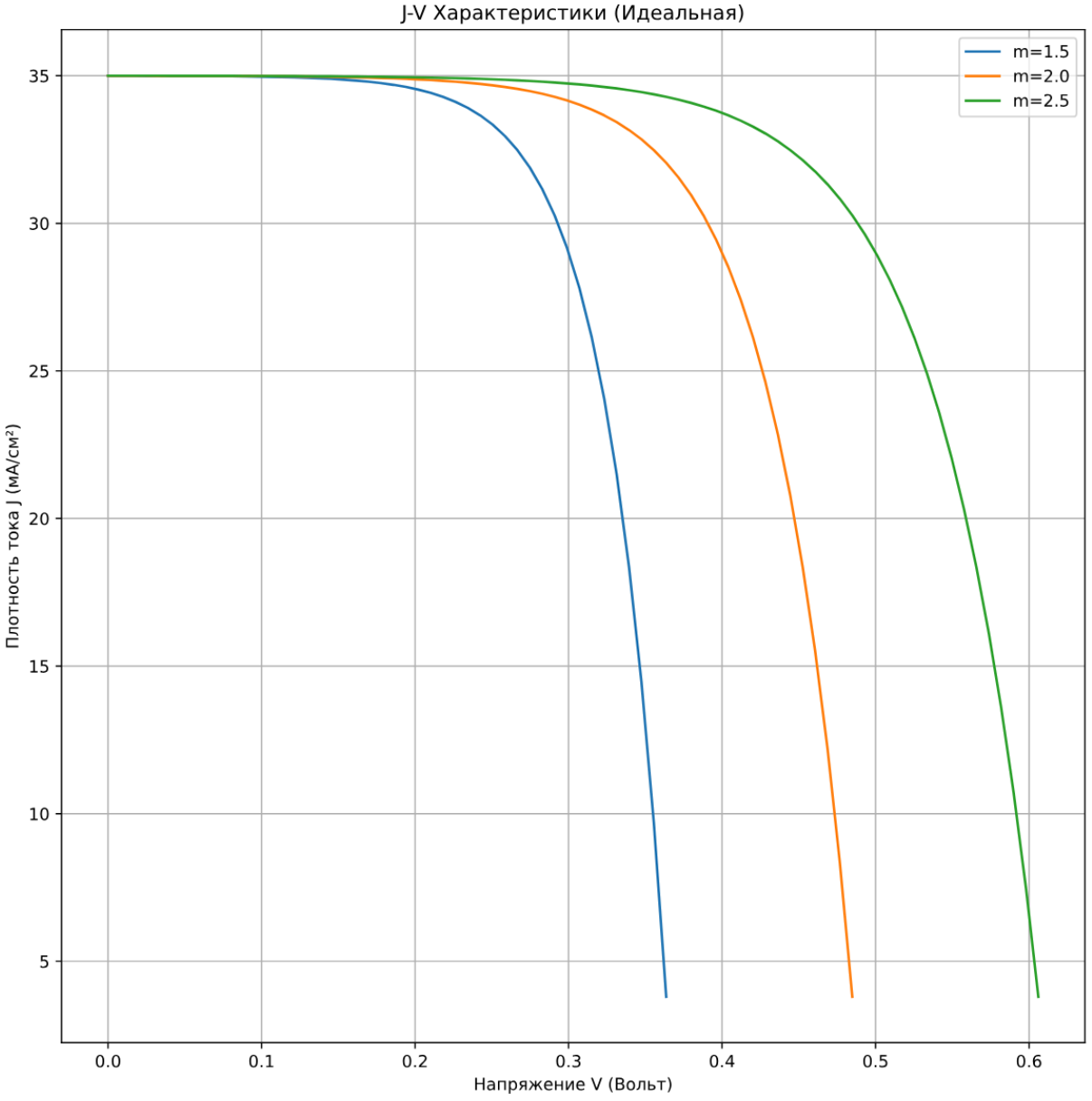
Код, для реализации пунктов 1.1 и 1.2

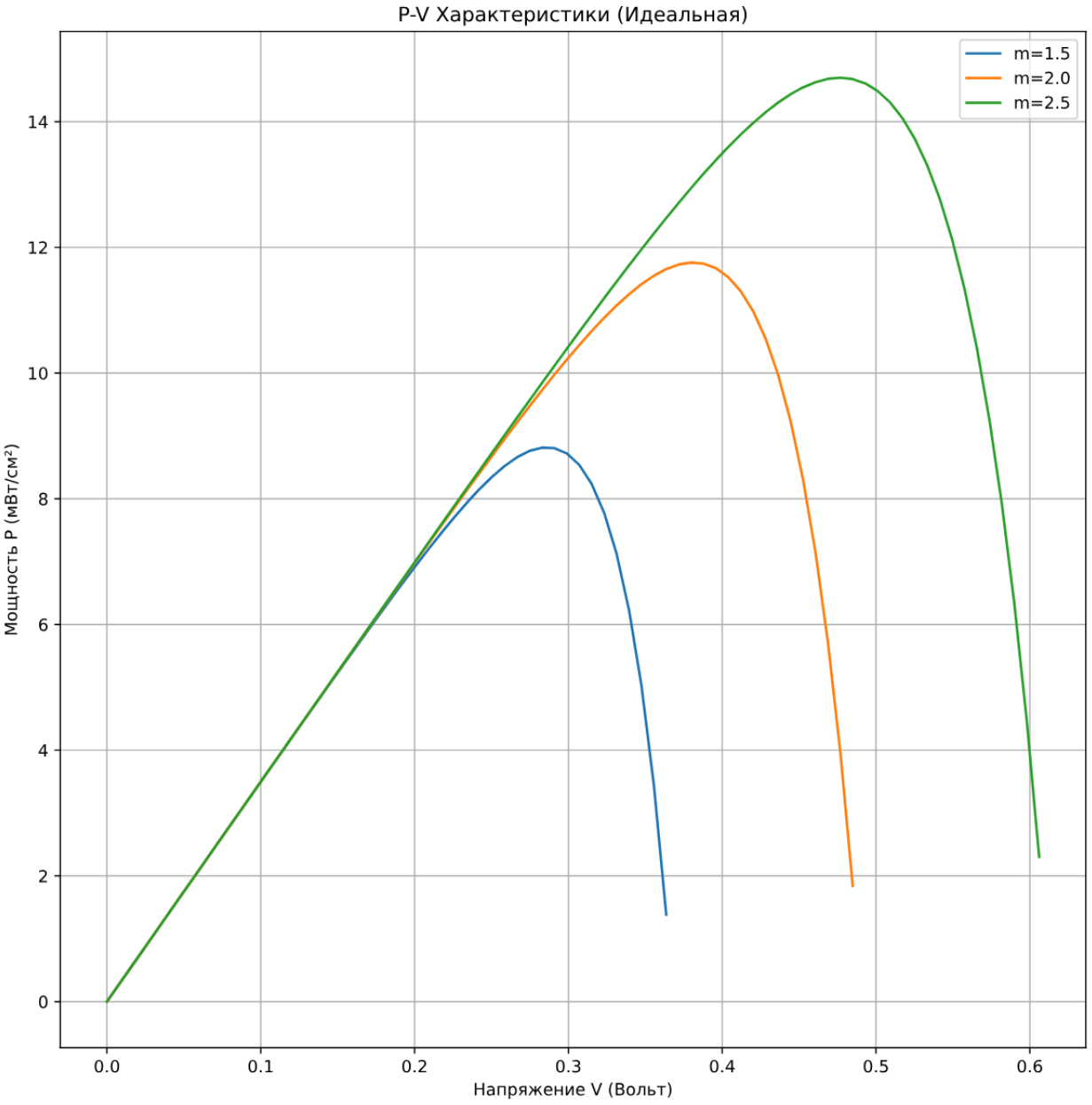
```
# КОНСТАНТЫ И ПАРАМЕТРЫ
e = 1.6e-19  # Элементарный заряд (C)
kB = 1.38e-23  # Постоянная Больцмана (J/K)
T = 298  # Температура (K), 25 C
A = 100  # Площадь ячейки (cm^2)

# Параметры ячейки
J_sc = 0.035  # Ток КЗ (A/cm^2) -> 35 mA/cm^2 (среднее из [25-45])
J_0 = 2.5e-6  # Ток насыщения (A/cm^2) -> 0.0025 mA/cm^2

# Термическое напряжение Vt = kBT / e
Vt = (kB * T) / e

# ЧАСТЬ 1: ИДЕАЛЬНАЯ ЯЧЕЙКА
def ideal_solar_cell(V, m, J_sc, J_0):
    """Уравнение (1): J = Jsc - J0 * (exp(V / (m*Vt)) - 1)"""
    # Защита от переполнения экспоненты
    arg = V / (m * Vt)
    arg = np.clip(arg, -100, 100)
    return J_sc - J_0 * (np.exp(arg) - 1)

# Диапазон напряжений для графика
V_ideal = np.linspace(0, 0.8, 100)

# 1. Построение графиков для разных m
```

```python
# m_values = [1.5, 2.0, 2.5]
m_values = np.linspace(1.5, 2.5, 3)
plt.figure(figsize=(12, 5))

# График J-V
plt.subplot(1, 2, 1)
for m in m_values:
    J_vals = ideal_solar_cell(V_ideal, m, J_sc, J_0)
    # Отфильтруем отрицательные токи для красоты графика
    mask = J_vals >= 0
    plt.plot(V_ideal[mask], J_vals[mask] * 1000, label=f"m={m}")  # J в
mA/cm^2

plt.title("J-V Характеристики (Идеальная)")
plt.xlabel("Напряжение V (Вольт)")
plt.ylabel("Плотность тока J (мА/см²)")
plt.grid(True)
plt.legend()

# График P-V
plt.subplot(1, 2, 2)
for m in m_values:
    J_vals = ideal_solar_cell(V_ideal, m, J_sc, J_0)
    P_vals = J_vals * V_ideal * 1000  # Мощность в мВт/см^2

    # Находим макс мощность
    p_max = np.max(P_vals)
    v_at_pmax = V_ideal[np.argmax(P_vals)]

    # Расчет Voc (теоретический)
    voc_theory = (m * Vt) * np.log((J_sc / J_0) + 1)

    print(f"m={m}: P_max={p_max:.2f} mW/cm^2, Voc={voc_theory:.3f} V")

    mask = J_vals >= 0
    plt.plot(V_ideal[mask], P_vals[mask], label=f"m={m}")

plt.title("P-V Характеристики (Идеальная)")
plt.xlabel("Напряжение V (Вольт)")
plt.ylabel("Мощность P (мВт/см²)")
plt.grid(True)
plt.legend()
```

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

Где полученный график выглядит так:





Объяснения результата

Можно заметить, что согласно графику, при увеличении m, максимальная достигаемая мощность растет. Это связано с тем, что математически формализуется модель идеализированного заторможенного диода. Это связано с тем, что согласно формуле $V_{oc} \approx \frac{mk_BT}{e} \ln\left(\frac{J_{sc}}{J_0}\right)$, V_{oc} растет пропорционально m, хотя в реальности, при увеличении фактора идеальности, мощность должна быть меньше, так как используется диод худшего качества. Этот математический артефакт объясняется использованием константным J_0, который в реальности был бы кратно больше, так как использовался бы дефектный кристалл. То есть в этой конкретной модели увеличивается сопротивляемость рекомбинации, не увеличивая саму рекомбинацию, что в реальности не может быть осуществлено.

Часть 2. Реальная модель

В реальной модели добавляются паразитные сопротивления: последовательное (R_s) и параллельное/шунтирующее (R_{sh}).

Уравнение, учитывающее паразитные сопротивления

Будет выглядеть следующим образом:

$$J = J_{sc} - J_0 \left(\exp\left(\frac{e(V + JAR_s)}{mk_BT} \right) - 1 \right) - \frac{V + JAR_s}{R_{sh}A}$$

Где:

- $\frac{V+JAR_s}{R_{sh}A}$ - плотность тока утечки, протекающего через шунтирующее сопротивление параллельно диоду.
- - $J \cdot A \cdot R_s$ - падение напряжения внутри солнечного элемента на его последовательном сопротивлении.
- A (Площадь перехода) - общая площадь поверхности p-n перехода солнечного элемента, используемая для пересчета плотности тока в абсолютное значение тока ($I = J \times A$).
- R_s (Последовательное сопротивление) - паразитное сопротивление контактов, металлических шин и самого полупроводника, которое приводит к падению выходного напряжения.
- R_{sh} (Шунтирующее сопротивление) - параллельное сопротивление, возникающее из-за утечек тока по краям устройства или через микродефекты, что снижает выходной ток.

Определение V_{oc} от R_{sh}

Для определения зависимости V_{oc} от R_{sh}:

1. Снова полагаем $J = 0$.
2. Тогда слагаемые с R_s исчезают, так как $J \cdot R_s = 0$.
3. Уравнение превращается в:

$$0 = J_{sc} - I_{diode} - I_{shunt}$$

$$0 = J_{sc} - J_0 \left(\exp\left(\frac{eV_{oc}}{mk_BT} \right) - 1 \right) - \frac{V_{oc}}{R_{sh}A}$$

4. Получается равенство, где J_{sc} (генерируемый ток) расходуется на два канала утечки: через диод и через сопротивление R_{sh}.

Численное решение (Метод Ньютона-Рафсона)

Поскольку уравнение нельзя решить аналитически (просто выразить J), нужно использовать численный метод.

Суть метода Ньютона:

Ищется J, при котором функция $f(J) = 0$. Уравнение, у которого все с одной стороны, будет выглядет так:

$$f(J) = J - J_{sc} + J_0(\dots) + \frac{V + JAR_s}{R_{sh}A} = 0$$

Алгоритм (реализован в функции `solve_real_current_newton`):

1. Предполагается, что ток J такой же, как в идеальной ячейке (без сопротивлений).
2. Уточнение: Мы вычисляем новое значение тока по формуле:

$$J_{new} = J_{old} - \frac{f(J_{old})}{f'(J_{old})}$$

   - Где $f'(J)$ — это производная функции по току.
3. Повторение: Повторяем шаг 2 несколько раз (в коде цикл `for _ in range(max_iter)`), пока разница между J_{new} и J_{old} не станет ничтожной (меньше `tol=1e-6`).

Код для вычислений, построения графиков и анализа

Код

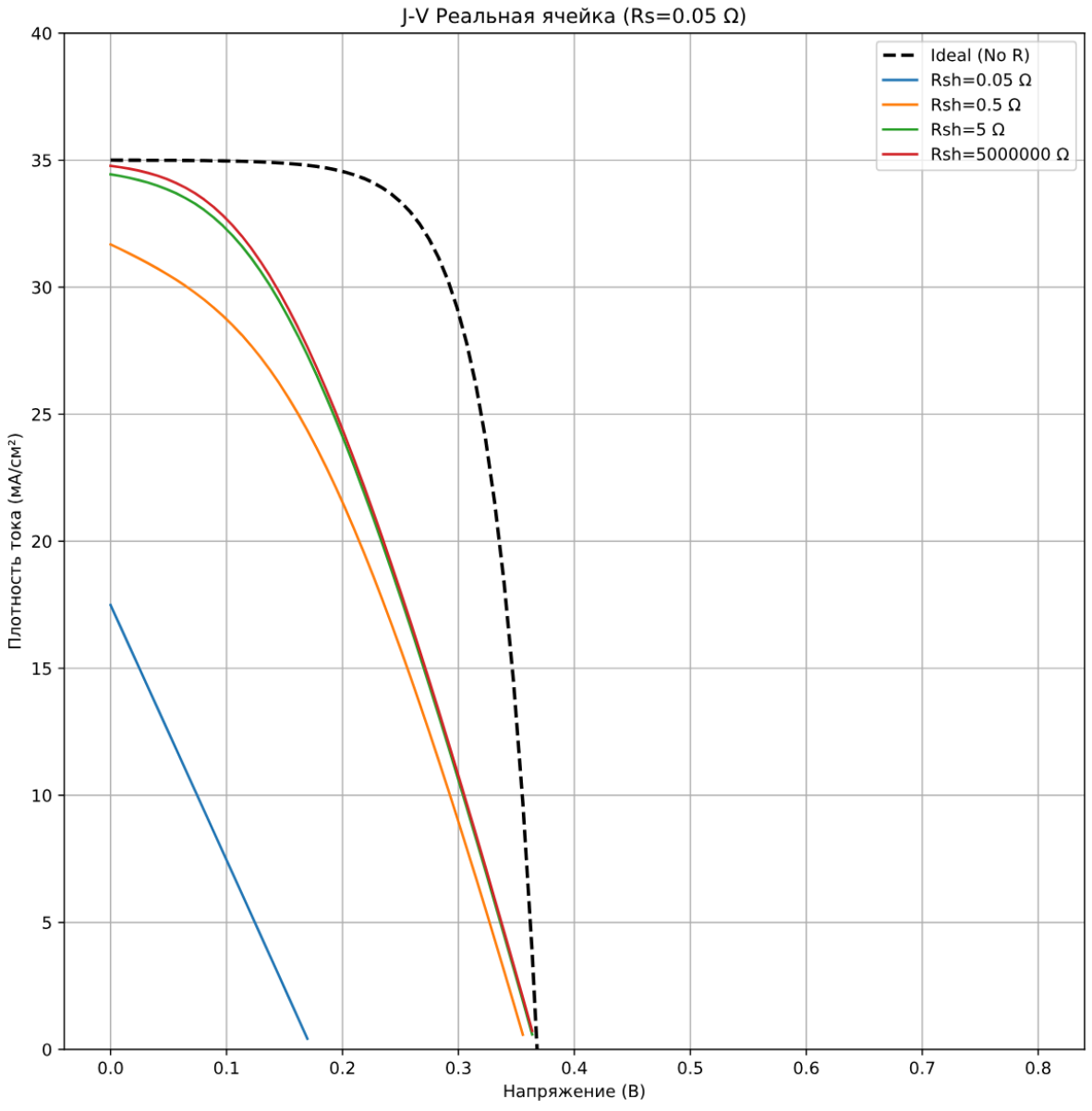
```
# ЧАСТЬ 2: РЕАЛЬНАЯ ЯЧЕЙКА (Группа МЭН)

# Параметры
m_real = 1.5  # Фиксируем m для сравнения сопротивлений
Rs = 0.05  # Rs = 50 mOhm = 0.05 Ohm (для площади 100cm2)

# Rsh должен быть минимум в 50 раз больше Rs, т.е. > 2.5 Ohm.
Rsh_values = [5, 20, 100, 1000]  # Ом

Rsh_values = [0.05, 0.5, 5, 5000000]


def solve_real_current_newton(V, Rs, Rsh, m, J_sc, J_0, A, tol=1e-6,
max_iter=50):
    """
    Решение уравнения (2) методом Ньютона-Рафсона для нахождения J при
заданном V.
    J выражается в A/cm^2.
    """
    # Начальное приближение: ток идеальной ячейки
```

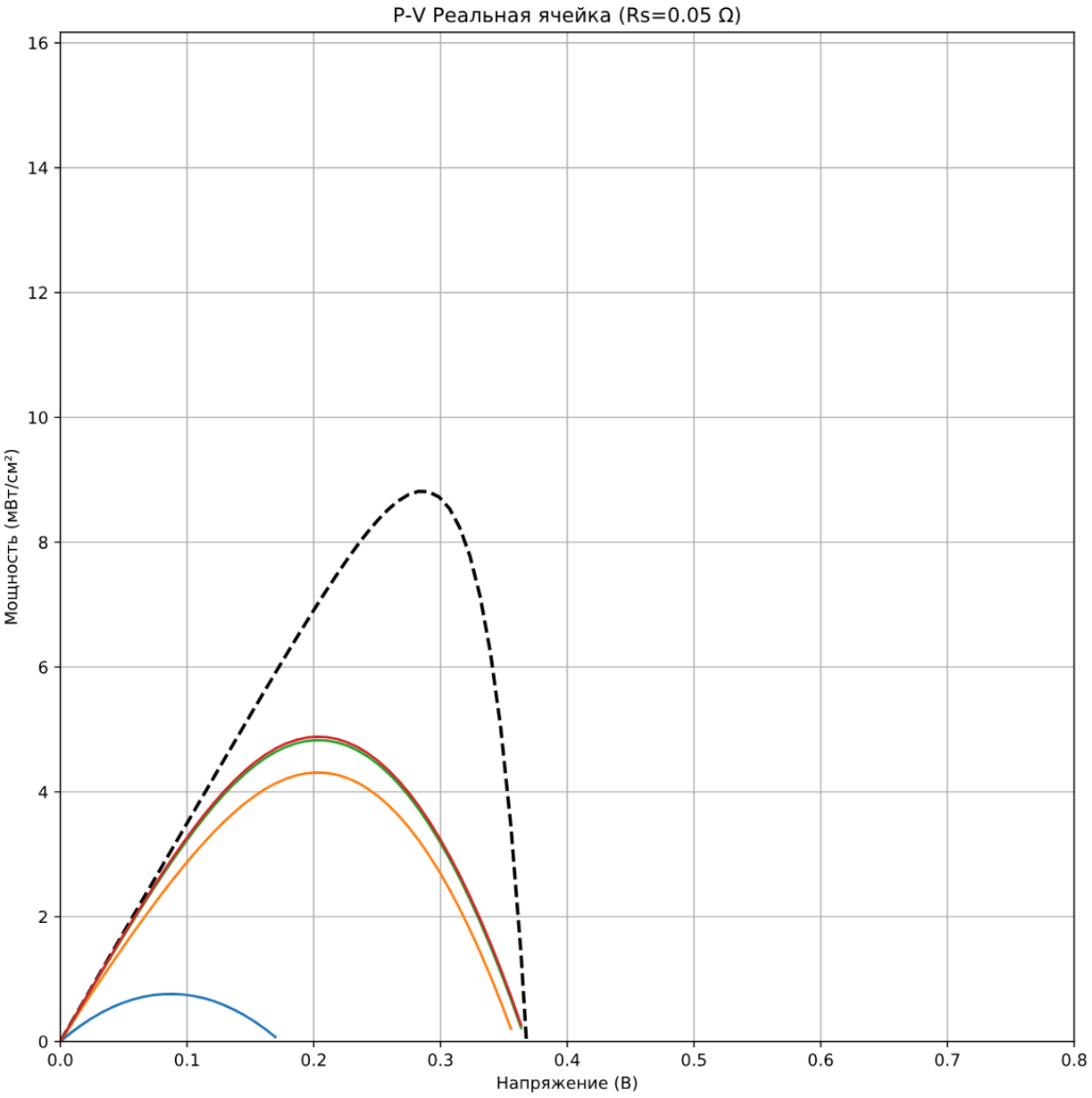
```
    J = ideal_solar_cell(V, m, J_sc, J_0)

    for _ in range(max_iter):
        # Вспомогательные переменные
        term_exp = np.exp((e * (V + J * A * Rs)) / (m * kB * T))

        # Функция f(J) = 0
        f_val = J - J_sc + J_0 * (term_exp - 1) + (V + J * A * Rs) / (Rsh
* A)

        # Производная f'(J)
        df_val = 1 + J_0 * (e * A * Rs / (m * kB * T)) * term_exp + (Rs *
A) / (Rsh * A)

        # Шаг Ньютона
        delta = f_val / df_val
        J_new = J - delta

        if abs(J_new - J) < tol:
            return J_new
        J = J_new

    return J  # Возвращаем последнее значение, если не сошлось (или для
области пробоя)

# Подготовка графиков
plt.figure(figsize=(12, 5))

# Сравнение с идеальной ячейкой (Rsh -> infinity, Rs -> 0)
V_range = np.linspace(0, 0.8, 100)
J_ideal = ideal_solar_cell(V_range, m_real, J_sc, J_0)
plt.subplot(1, 2, 1)
plt.plot(V_range, J_ideal * 1000, "k--", linewidth=2, label="Ideal (No
R)")
plt.subplot(1, 2, 2)
plt.plot(V_range, J_ideal * V_range * 1000, "k--", linewidth=2,
label="Ideal (No R)")

# Цикл по значениям Rsh
for Rsh in Rsh_values:
    J_real_curve = []
    for V in V_range:
        j_point = solve_real_current_newton(V, Rs, Rsh, m_real, J_sc, J_0,
```

```
A)
        J_real_curve.append(j_point)
    J_real_curve = np.array(J_real_curve)
    P_real_curve = J_real_curve * V_range * 1000  # mW/cm^2

    # Фильтрация (J > 0) для отображения первого квадранта
    mask = J_real_curve >= 0
    # График J-V
    plt.subplot(1, 2, 1)
    plt.plot(V_range[mask], J_real_curve[mask] * 1000, label=f"Rsh={Rsh}
$\Omega$")

    # График P-V
    plt.subplot(1, 2, 2)
    plt.plot(V_range[mask], P_real_curve[mask], label=f"Rsh={Rsh}
$\Omega$")

    # Вывод максимумов
    if np.any(mask):
        p_max = np.max(P_real_curve[mask])
        print(f"Rsh={Rsh} Ohm: P_max={p_max:.2f} mW/cm^2")

# Настройка осей для Части 2
plt.subplot(1, 2, 1)
plt.title(f"J-V Реальная ячейка (Rs={Rs} $\Omega$)")
plt.xlabel("Напряжение (В)")
plt.ylabel("Плотность тока (мА/см²)")
plt.grid(True)
plt.legend()
plt.ylim(0, J_sc * 1000 + 5)

p_limit = 14.70 * 1.1

plt.subplot(1, 2, 2)
plt.title(f"P-V Реальная ячейка (Rs={Rs} $\Omega$)")
plt.xlabel("Напряжение (В)")
plt.ylabel("Мощность (мВт/см²)")
plt.grid(True)

# Устанавливаем лимиты:
plt.xlim(0, 0.8)  # Напряжение от 0 до 0.8 В (охватывает все Voc из
задания)
plt.ylim(0, p_limit)
```

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

Графики





Объяснение работы

В коде происходит следующее:

1. Берется список значений R_{sh} (0.05, 0.5, 5, 5000000 Ом). Чем меньше число, тем хуже ячейка (больше утечек).
2. Для каждого R_{sh} код итерируется по всему диапазону напряжений от 0 до 0.8 Вольт.
3. В каждой точке напряжения V мы запускается метод Ньютона (Шаг 3), чтобы найти точный ток J.
4. Считается мощность $P = J \cdot V$
5. Вывод из графиков:
   - При **малом** R_{sh} (5 Ом) график тока резко идет вниз. Это потому, что ток предпочитает течь через легкий путь (шунт), а не во внешнюю цепь. P_{max} резко

падает.

- При **большом** R_{sh} (1000 Ом) сопротивление утечки велико, ток туда не идет, и ячейка работает почти как идеальная.

Таким образом, решение показывает, как ненадежная в изоляция (низкое шунтирующее сопротивление) убивают эффективность солнечной панели.

Исправления

Исследование сходимостей метода Ньютона-Рафсона и метода простых итераций

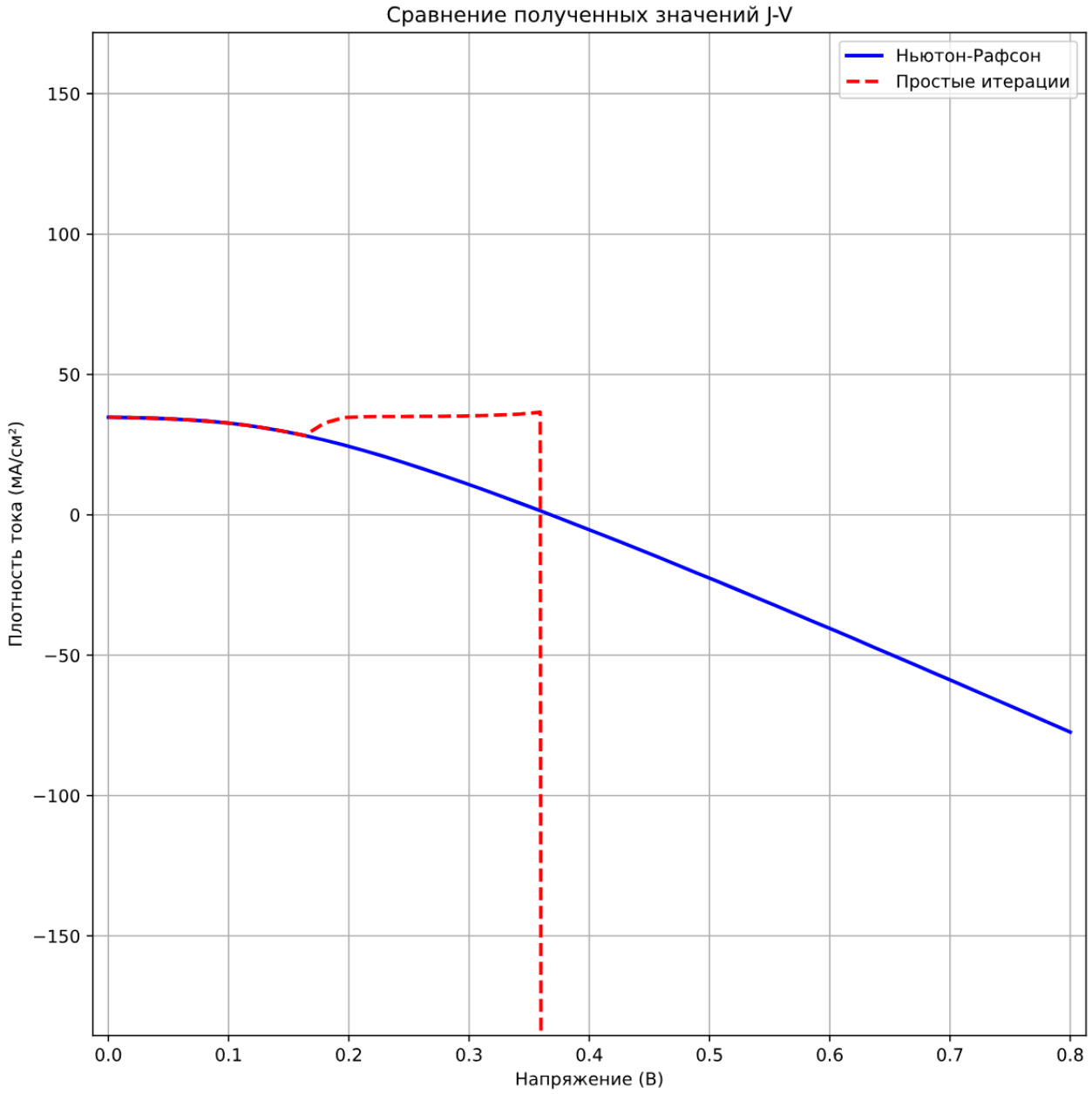
Код

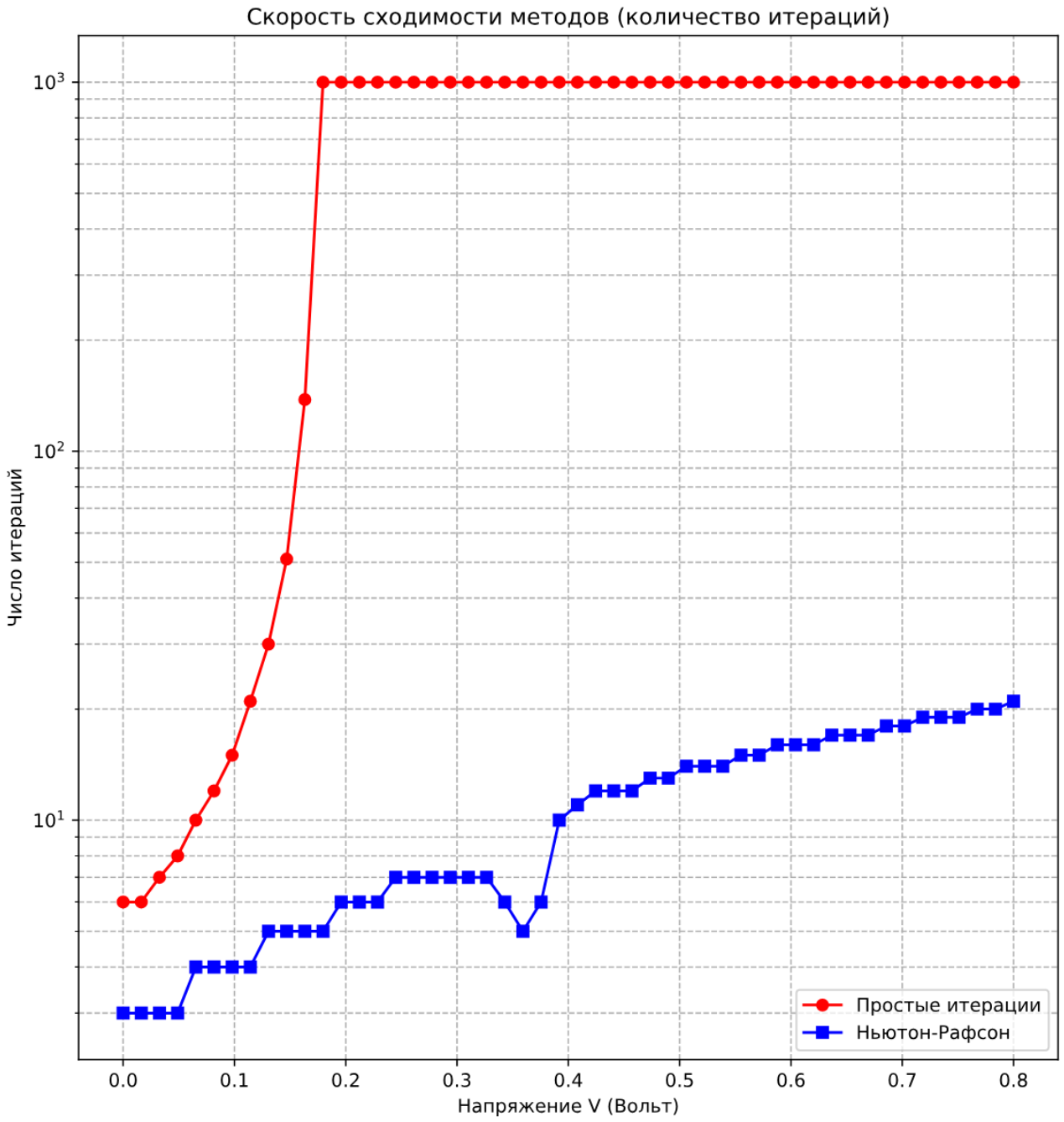
```
import numpy as np
import matplotlib.pyplot as plt

# Параметры ячейки
e = 1.6e-19
kB = 1.38e-23
T = 298
Vt = (kB * T) / e
A = 100
m = 1.5
Jsc = 0.035
J0 = 2.5e-6
Rs = 0.05  # 50 mOhm
Rsh = 100  # Шунт


# МЕТОД ПРОСТЫХ ИТЕРАЦИЙ
def simple_iteration(V, tol=1e-10):
    # Начальное приближение: идеальный ток (Eq. 1)
    J = Jsc - J0 * (np.exp(V / (m * Vt)) - 1)
    iters = 0
    for i in range(1000):
        # Вычисляем правую часть уравнения (2)
        J_next = (
            Jsc
            - J0 * (np.exp((e * (V + J * A * Rs)) / (m * kB * T)) - 1)
            - (V + J * A * Rs) / (Rsh * A)
        )
        iters += 1
```

```
        if abs(J_next - J) < tol:
            return J_next, iters
        J = J_next
    return J, iters


# 2. МЕТОД НЬЮТОНА-РАФСОНА (Newton-Raphson)
def newton_raphson(V, tol=1e-10):
    J = Jsc - J0 * (np.exp(V / (m * Vt)) - 1)
    iters = 0
    for i in range(100):
        arg = (e * (V + J * A * Rs)) / (m * kB * T)
        exp_term = np.exp(arg)
        # f(J) = 0
        f = J - Jsc + J0 * (exp_term - 1) + (V + J * A * Rs) / (Rsh * A)
        # Производная f'(J)
        df = 1 + J0 * (e * A * Rs / (m * kB * T)) * exp_term + Rs / Rsh

        J_next = J - f / df
        iters += 1
        if abs(J_next - J) < tol:
            return J_next, iters
        J = J_next
    return J, iters


# Сбор данных для сравнения
V_vals = np.linspace(0, 0.8, 50)
data_newton = [newton_raphson(v) for v in V_vals]
data_simple = [simple_iteration(v) for v in V_vals]

# Построение графика сравнения скорости
# plt.figure(figsize=(10, 5))
plt.plot(V_vals, [d[1] for d in data_simple], "r-o", label="Простые
итерации")
plt.plot(V_vals, [d[1] for d in data_newton], "b-s", label="Ньютон-
Рафсон")
plt.yscale("log")  # Логарифмическая шкала
plt.title("Скорость сходимости методов (количество итераций)")
plt.xlabel("Напряжение V (Вольт)")
plt.ylabel("Число итераций")
plt.grid(True, which="both", ls="--")
plt.legend()
```

```
plt.show()


J_newton = np.array([d[0] for d in data_newton])
J_simple = np.array([d[0] for d in data_simple])

# --- ПОСТРОЕНИЕ ГРАФИКОВ ---

# График 1: Сравнение J-V кривых
plt.plot(V_vals, J_newton * 1000, "b-", linewidth=2, label="Ньютон-
Рафсон")
plt.plot(V_vals, J_simple * 1000, "r--", linewidth=2, label="Простые
итерации")
plt.title("Сравнение полученных значений J-V")
plt.xlabel("Напряжение (В)")
plt.ylabel("Плотность тока (мА/см²)")
plt.legend()
plt.grid(True)


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

Графики


Сравнение полученных значений J-V
Ньютон-Рафсон
Простые итерации
Плотность тока (мА/см²)
Напряжение (В)
150
100
50
0
−50
−100
−150
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8



Выводы:

Анализ графиков сравнения полученных значений $J - V$ показывает принципиальную разницу в стабильности алгоритмов:

- Метод Ньютона-Рафсона (синяя линия): Демонстрирует высокую стабильность во всем диапазоне напряжений (от 0 до 0.8 В). Он корректно находит значения тока даже в области отрицательных значений, что важно для полного анализа вольтамперной характеристики.

- Метод простых итераций (красная пунктирная линия): Совпадает с методом Ньютона только на начальном участке (до ≈ 0.18 В). После этой точки метод перестает сходиться к истинному значению, демонстрируя резкий скачок и последующий обрыв при $V \approx 0.36$ В. Это происходит из-за того, что при определенных напряжениях производная итерационной функции становится больше единицы.

График количества итераций наглядно иллюстрирует преимущество метода Ньютона в эффективности:

- Ньютон-Рафсон: Требует минимального количества вычислений во всем диапазоне. Даже при высоких напряжениях, где сложность задачи растет, число шагов увеличивается незначительно.
- Простые итерации: До 0.1 В метод работает относительно быстро (менее 10 итераций), но затем число шагов экспоненциально растет. При напряжении выше 0.18 В алгоритм достигает лимита в 1000 итераций (при изменении лимита, алгоритм так же в него упрется), так и не найдя точного решения.

Определение V_{oc} как функции от R_{sh}

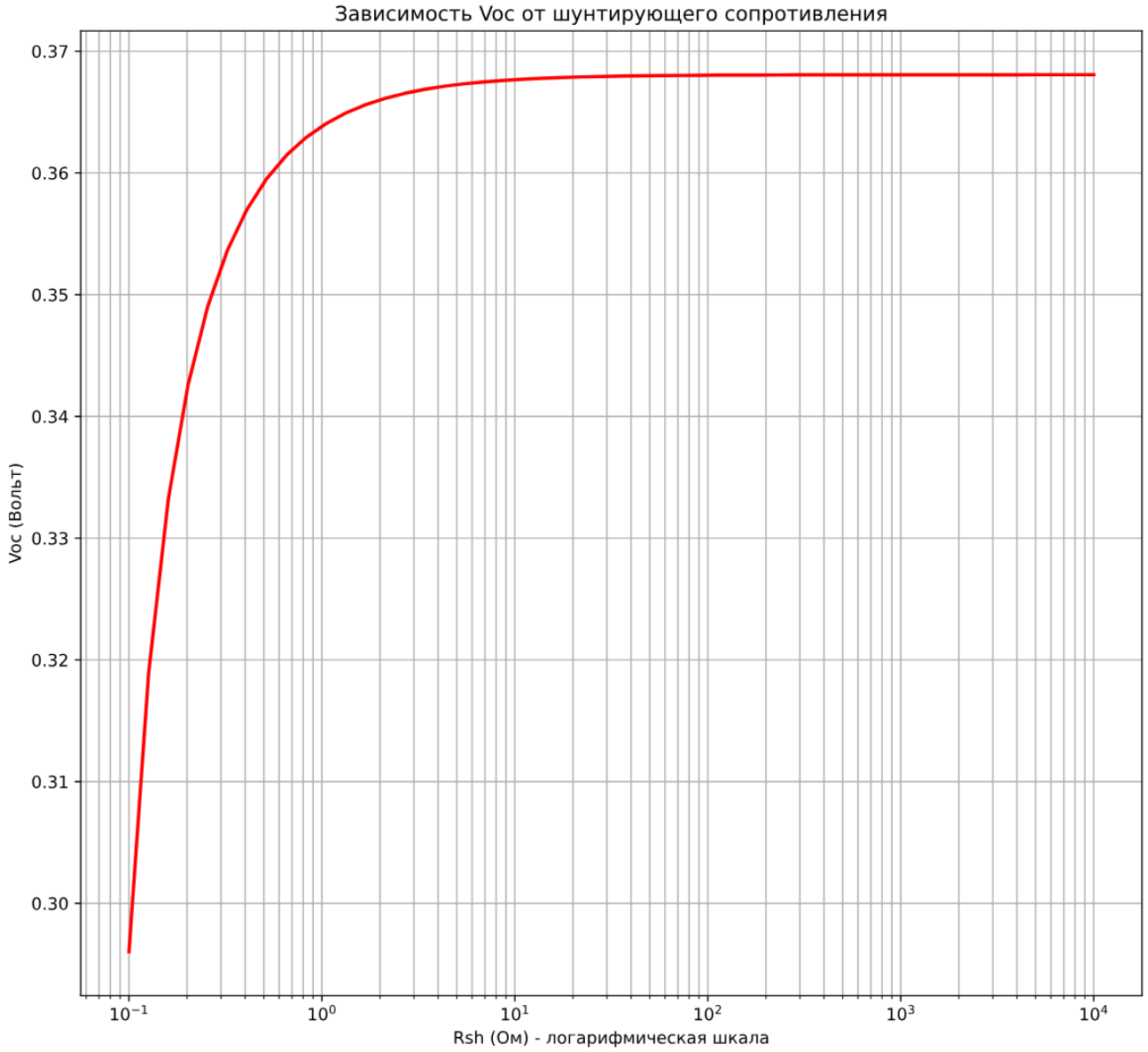
Код

```
import numpy as np
import matplotlib.pyplot as plt

# --- КОНСТАНТЫ И ПАРАМЕТРЫ ---
e = 1.6e-19  # Элементарный заряд (Кл)
kB = 1.38e-23  # Постоянная Больцмана (Дж/К)
T = 298  # Температура (К)
A = 100  # Площадь ячейки (см^2)
J_sc = 0.035  # Ток КЗ (А/см^2)
J_0 = 2.5e-6  # Ток насыщения (А/см^2)
m = 1.5  # Фактор идеальности
Rs = 0.05  # Фиксированное Rs (Ом)
Vt = (kB * T) / e  # Термическое напряжение


# 1. Функция для поиска тока J методом Ньютона-Рафсона
def solve_J(V, Rsh):
    J = J_sc  # Начальное приближение
    for _ in range(50):
        # Вспомогательная экспонента
        arg = (e * (V + J * A * Rs)) / (m * kB * T)
        arg = np.clip(arg, -100, 100)  # Защита от переполнения
```

```
        term_exp = np.exp(arg)

        # Функция f(J) = 0
        f = J - J_sc + J_0 * (term_exp - 1) + (V + J * A * Rs) / (Rsh * A)
        # Производная f'(J)
        df = 1 + J_0 * (e * A * Rs / (m * kB * T)) * term_exp + Rs / Rsh

        J_new = J - f / df
        if abs(J_new - J) < 1e-7:
            return J_new
        J = J_new
    return J


# 2. Функция для поиска Voc (когда J=0)
def find_voc(Rsh):
    # При J=0 уравнение: J_sc - J_0(exp(e*V/(m*kB*T))-1) - V/(Rsh*A) = 0
    V = 0.3  # Начальное приближение
    for _ in range(50):
        arg = (e * V) / (m * kB * T)
        term_exp = np.exp(arg)
        f = J_sc - J_0 * (term_exp - 1) - V / (Rsh * A)
        df = -J_0 * (e / (m * kB * T)) * term_exp - 1 / (Rsh * A)
        V_new = V - f / df
        if abs(V_new - V) < 1e-7:
            return V_new
        V = V_new
    return V


# --- ПОСТРОЕНИЕ ГРАФИКОВ ---
V_range = np.linspace(0, 0.5, 100)
plt.figure(figsize=(12, 5))

Rsh_axis = np.logspace(-1, 4, 50)
Voc_vals = [find_voc(r) for r in Rsh_axis]

plt.semilogx(Rsh_axis, Voc_vals, color="red", linewidth=2)
plt.title("Зависимость Voc от шунтирующего сопротивления")
plt.xlabel("Rsh (Ом) - логарифмическая шкала")
plt.ylabel("Voc (Вольт)")
plt.grid(True, which="both", ls="-")
```

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

График



Выводы

На графике отчетливо видна область резкого падения напряжения при значениях $R_{sh} < 10$ Ом:

- При малых значениях шунтирующего сопротивления (когда изоляция ячейки нарушена дефектами или плохой обработкой краев) ток утечки становится сопоставим с фотогенерированным током J_{sc}.

- При увеличении R_{sh} выше 10^2 (100) Ом график выходит на горизонтальное плато. При достаточно высоком сопротивлении утечка через шунт становится ничтожной по сравнению с током через диод. Начиная с этого момента, V_{oc} практически перестает зависеть от R_{sh} и определяется параметрами полупроводника (J_{sc}, J_0, m).